UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

Geração Automática de Montadores a Partir de ArchC: Um Estudo de Caso com o PowerPC 405

AUTORES:
Daniel Carlos Casarotto
José Otávio Carlomagno Filho

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

ORIENTADOR:
Prof. Luiz Cláudio Villar dos Santos

Florianópolis, SC
2004/2

Daniel Carlos Casarotto
José Otávio Carlomagno Filho

Geração Automática de Montadores a Partir de ArchC: Um Estudo de Caso com o PowerPC 405

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação.

_____
Orientador: Prof. Luiz Cláudio Villar dos Santos

Banca Examinadora

_____
Prof. Olinto José Varela Furtado

_____
Prof. Luís Fernando Friedrich

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Os sistemas computacionais evoluem de forma cada vez mais rápida, sendo hoje possível a criação de sistemas inteiros, compostos por CPUs, memória e periféricos, dentro de uma única pastilha de silício, os chamados *System-on-Chip* (SoCs). Estes sistemas podem incorporar CPUs de propósitos gerais ou CPUs projetadas especificamente para uma aplicação (os ASIPs). Se por um lado os ASIPs permitem uma grande otimização dos SoCs pois representam um compromisso entre flexibilidade, desempenho e eficiência no consumo de energia, por outro lado eles não são componentes padrão, desta forma não estão previamente disponíveis para eles importantes ferramentas de suporte à geração de código, como compiladores e montadores. Para acomodar a pressão do chamado *time-to-market*, ou seja, colocar o produto no mercado o quanto antes, torna-se necessária a automatização do processo de geração destas ferramentas, pois a sua geração manual inviabilizaria o uso dos ASIPs. Este trabalho apresenta técnicas para a geração automática de montadores e pré-processadores a partir de descrições feitas em uma linguagem de descrição de arquiteturas (o ArchC). Como estudo de caso, um modelo do processador PowerPC 405 foi desenvolvido utilizando-se o ArchC e validado através da execução de *benchmarks*, e este modelo foi também utilizado em experimentos com o gerador de montadores, juntamente com os modelos ArchC dos processadores MIPS e do PIC 16F84.

PALAVRAS CHAVE: ADLs, ASIPs, gramáticas, montadores, compiladores

Capítulo 1

Introdução

A tecnologia dos sistemas computacionais evolui de forma rápida ao longo dos anos. O principal parâmetro utilizado para medir e prever esta evolução é a chamada *Lei de Moore*, a qual diz que a cada 18 meses duplica o número de transistores que podem ser inseridos em uma mesma pastilha de silício.

Cada vez mais os sistemas computacionais estão sendo utilizados em diferentes lugares além dos computadores de propósito geral (computadores pessoais, servidores, etc). Atualmente, não é difícil encontrarmos microprocessadores em eletrodomésticos, automóveis, telefones celulares, sistemas de controle de vôo em aviões, sistemas de automação industrial, entre outros.

Estes sistemas, que normalmente incluem, além de um micro-processador, sensores, conversores analógico-digital e memória e que normalmente são específicos para uma dada aplicação, são conhecidos como sistemas embarcados (ou embutidos). Atualmente, 79% dos processadores de alto desempenho no mercado são usados em sistemas embutidos [2]. Segundo Marwedel [2], os modelos mais avançados da marca BMW contêm mais de 100 microprocessadores.

No início, os sistemas embarcados eram construídos em placas de circuito impresso. Em seguida, tornou-se possível agregar alguns componentes dentro de uma mesma pastilha de silício, dando origem à metodologia de *System in a Package* (SiP). Após isso, a evolução tecnológica permitiu que se pudesse integrar todo o sistema em uma única pastilha de silício, que são os denominados *SoCs (System on a Chip)*. Os processadores convencionais são voltados para o mercado de desktops e servidores, onde é necessário atender uma ampla gama de aplicações, para isso é necessário que tenham recursos suficientes. O que acontece neste caso é que, muitas vezes boa parte destes recursos ficam ociosos, ou não são utilizados de maneira eficiente, tornando-os

ineficientes como processadores para uso específico. O processador *PowerPC 405*, que é voltado para o mercado de sistemas embarcados e que será o estudo de caso deste trabalho, por exemplo, não possui unidade de ponto flutuante, pois é voltado para aplicações que não necessitam de operações em ponto flutuante ou casos em que a emulação destas operações é suficiente.

Para tentar contornar essa situação, surgiram outras alternativas de projeto, dentre elas os *ASIPs* (*Application Specific Instruction-set Processor*), que são processadores com um conjunto de instruções escolhidos especialmente para um conjunto de aplicações. Como os *ASIPs* são desenvolvidos dependendo das aplicações e, portanto, não possuem softwares nem ferramentas de desenvolvimento de softwares previamente construídas e prontas para o uso, surge a necessidade da automatização da geração de ferramentas de suporte ao desenvolvimento de software, que é o foco do gerador automático de montadores.

A implementação e depuração das ferramentas, bem como os resultados experimentais obtidos nesta monografia foram objeto de trabalho cooperativo entre os autores e o aluno de mestrado Leonardo Taglietti conforme artigo cientifico escrito em co-autoria [11]. A elaboração e depuração do modelo do *PowerPC 405* foram executadas exclusivamente pelos autores deste trabalho.

1.1 Linguagens de Descrição de Arquiteturas

Linguagens de descrição de arquiteturas, ou simplesmente *ADLs (Architecture Description Languages)*, são linguagens que permitem a descrição das instruções de um processador assim como (em algumas destas linguagens) a descrição de outras características de uma aquitetura como memória, *cache*, *pipeline*, etc. Dentre essas linguagens destacam-se *LISA*, *EXPRESSION*, *nML*, *ISDL* e *ArchC*[3]. Esta última foi adotada para ser usada neste trabalho pois ainda não possui um gerador automático de montadores, embora exista um projeto com esse fim, além de ser a primeira a gerar simuladores em *SystemC*[4]. SystemC é uma extensão das bibliotecas padrão da linguagem C++, através da qual é possível descrever características de sistemas embarcados, tais como paralelismo do hardware, tempo, separação entre funcionalidade e comunicação, etc. SystemC é considerada a linguagem mais promissora para a modelagem de sistemas embutidos, especialmente nos mais altos níveis de abstração [10].

Dentre as características do *ArchC*[3] destacam-se a capacidade de descrição do conjunto de instruções, precisão de ciclos, suporte à multi-ciclo, *pipeline*, hierarquia de memória, emulação de sistema operacional, simulação compilada, dentre outras.

Com a crescente pressão do *time-to-market*, e com a impossibilidade de testar o software de um sistema embutido antes de ele estar pronto, a metodologia conhecida como integração software-hardware, que consiste no projeto do software após o projeto do hardware começou a perder espaço para a metologia do *HW/SW co-design*, ou seja, o hardware e o software são projetados ao mesmo tempo. Isso só tornou-se possível através dos modelos executáveis (simuladores) de hardware que podem ser gerados por uma *ADL*, por exemplo.

Capítulo 2

Modelagem do processador PowerPC 405

O processador *PowerPC 405* foi escolhido como objeto de estudo deste trabalho pois foi feito para ser usado no mercado de sistemas embutidos, possuindo desta forma, características que são descritas posteriormente que o diferenciam das demais versões do *PowerPC*. Ele tornou-se popular estando presente em várias plataformas como a *Virtex-II*, sendo usado desde câmeras de TV, roteadores, até sistemas de automação industrial. É um processador RISC (*Reduced Instruction Set Computer*) de 32 bits, cuja arquitetura, desenvolvida em conjunto pela Motorola, IBM e Apple Computer, é baseada na arquitetura POWER, implementada pela família de microprocessadores RS/6000. Essa arquitetura foi desenvolvida visando compatibilidade de software entre diversas famílias de implementações. É voltada para o mercado embutido, buscando alta performance e baixo consumo de energia. É um SoC que integra o processador (*core*) e vários periféricos em uma única pastilha de silício.

2.1 Arquitetura do conjunto de instruções

Dentre suas principais características destacam-se:

- Banco de registradores com 32 entradas
- Instruções para carregar e salvar dados entre registradores e memória e entre registradores. É uma arquitetura de uma máquina *load-store*
- Instruções de tamanho fixo, o que possibilita um melhor uso do *pipeline* e torna mais fácil a execução paralela de instruções

- Uso não destrutivo dos registradores, ou seja, as instruções aritméticas possuem um campo destino separado dos campos fonte
- Possui um modelo preciso de exceções
- Instruções para manipular explicitamente a memória *cache* (salvar, invalidar dados). Permite também que se carreguem dados para a *cache*, através de instruções, antes que eles sejam necessários, reduzindo assim a latência de memória
- Suporte tanto para cache de dados e instruções separadas (arquiterura *Harvard*) quanto para caches unificadas
- Define 32 bits, (4GB) de espaço endereçável para memória e periféricos

2.1.1 Registradores

GPR - (*General Purpose Registers*) Registradores de propósito geral são 32 ao total, através de instruções load é possível ler um valor da memória e colocá-lo nesses registradores, da mesma forma é possível gravar um valor contido em um registrador na memória através do uso de instruções do tipo store. A maior parte das instruções inteiras usam esse banco de registradores como operandos fonte e destino.

CTR - (*Count Register*) - Registrador de contagem é usado pelas instruções de desvio como contador, onde é decrementado e testado sob certas circunstâncias, pode ser usado para guardar um endereço que será usado pela instrução bcctr, possibilitando assim o desvio para qualquer posição de memória.

LR - (Link Register) - Registrador de ligação é usado principalmente para armazenar o endereço de retorno de sub-rotinas. Pode ser escrito por qualquer instrução de branch que tenha o bit LK como 1, desta forma será guardado o endereço da instrução seguinte à instrução de branch no registrador LR. A instrução bclr usa o conteúdo do LR como destino de seu desvio.

XER - (*Fixed Point Exception Register*) - Registrador de excessões em ponto fixo guarda informações de overflow e carry geradas pelas instruções aritméticas. Instruções da forma “o” e da forma “.”: Esses tipos de instruções possuem o mnemônico terminando com “o” ou “.”. As instruções do tipo “o” são as únicas além das instruções mtspr e mcrxr que podem modificar os

campos SO e OV do registrador XER. As instruções do tipo “.” modificam o primeiro campo do registrador CR. Além do mnemônico diferente, essas instruções possuem um bit em seu formato para informar se são do tipo “o” e outro bit para informar se são do tipo “.”. Exemplo:

- add - Modifica apenas seu registrador de destino.
- add. - Modifica também o campo “CR0” do registrador CR, indicando se o resultado da soma foi positivo, negativo ou igual a zero (ver CR)
- addo - Modifica os campos SO e OV do registrador XER se ocorreu overflow.
- addo. - Modifica tanto o registrador CR quanto o XER.

Esse registrador é dividido conforme a figura 2.1:

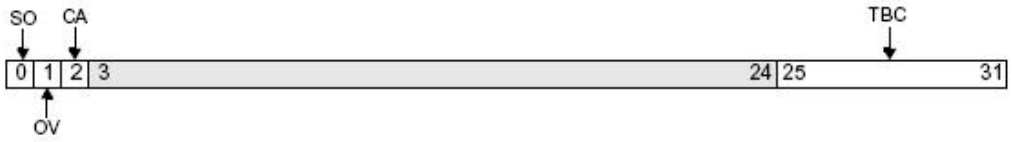

Figura 2.1: Fixed point exception register

O bit CA, é modificado por certas instruções aritméticas que indicam se aconteceu *carry out* ou não na operação.

CR - Condition Register contém 8 campos de 4 bits cada. Armazenam condições que são detectadas após a realização de certas instruções. As instruções de comparação podem mudar esse registrador, e a interpretação de seus bits é feita da seguinte forma:

- Bit 0 - LT - O primeiro operando é menor que o segundo operando da instrução.
- Bit 1 - GT - O primeiro operando é maior que o segundo operando da instrução.
- Bit 2 - EQ - Os dois operandos da instrução são iguais.
- Bit 2 - SO - Bit de overflow, é uma cópia do bit SO do registrador XER.

Instruções da forma “.” também podem mudar esse registrador, a forma de interpretar os bits fica um pouco diferente, neste caso é comparado o resultado da operação com 0, verificando se ele é menor (LT), maior (GT), igual (EQ) a zero.

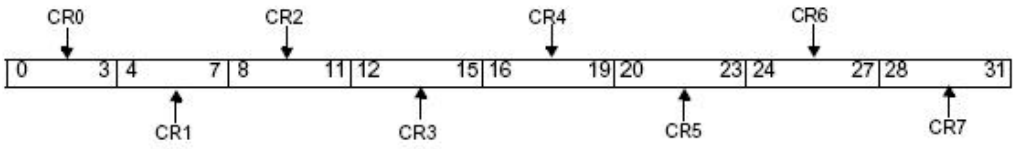

Figura 2.2: Condition register

2.1.2 Formato das instruções

Todas as instruções possuem 32 bits de tamanho, sendo que os primeiros 6 bits de todas as instruções possui o opcode. Algumas instruções possuem um opcode adicional em outro campo. Os formatos são divididos de acordo com as figuras abaixo, na primeira posição aparece a forma geral do formato, abaixo, estão algumas variantes do mesmo. Nem todas as variantes dos formato estão sendo mostradas, apenas as mais importantes, pois em algumas delas, a única diferença está apenas na nomenclatura dos campos.

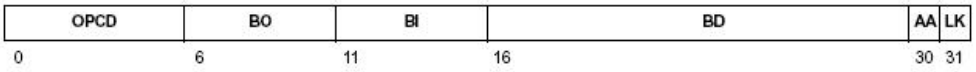

Figura 2.3: Formato B

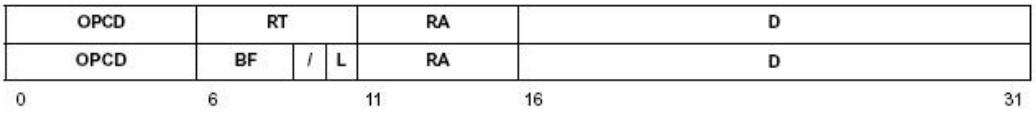

Figura 2.4: Formato D

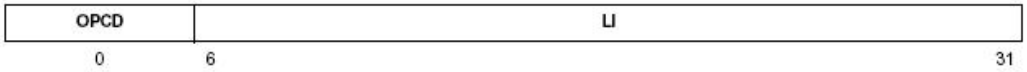

Figura 2.5: Formato I

As instruções podem ser divididas em grupos conforme sua utilização:

Instruções de armazenamento são basicamente as instruções de *load* que carregam um valor da memória em um registrador e as instruções de *store* que salvam um valor de um registrador na memória. Podem operar sobre byte, halfword, word e strings.

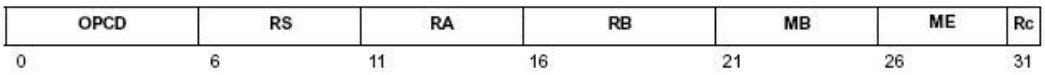

Figura 2.6: Formato M

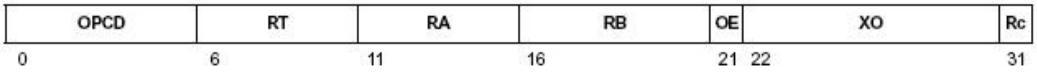

Figura 2.7: Formato XO

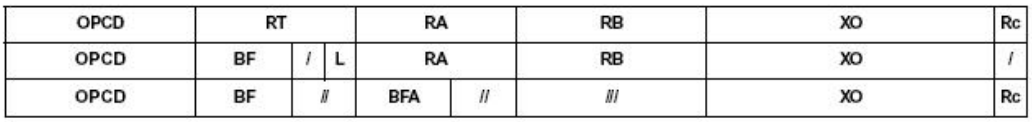

Figura 2.8: Formato X

Figura 2.9: Formato XFX

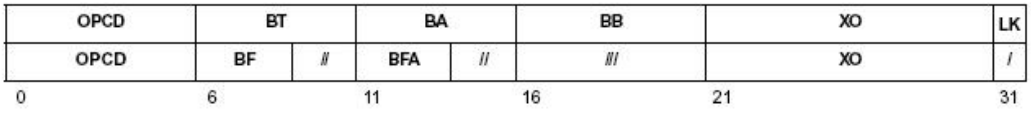

Figura 2.10: Formato XL

Loads				Stores			
Byte	Halfword	Word	String	Byte	Halfword	Word	String
lbz[u][x]	lha[u][x]	lwarx	lmw	stbu[u][x]	sth[u][x]	stw[u][x]	stmw
	lhz[u][x]	lwbrx	lswi		sthbrx	stwbrx	stswi
	lhbrx	lwz[u][x]	lswx			stwcx.	stswx

Tabela 2.1: Intruções de Load e Store

Instruções com o sufixo “u” atualizam um dos registradores fontes com o valor calculado do endereço. Instruções com o sufixo “x” calculam o endereço através da soma dos valores contidos nos dois registradores indicados em seus operandos. As outras calculam o endereço através da soma de um registrador com um campo imediato de 16 bits.

Instruções aritméticas executam operações aritméticas inteiras, executam alguma operação em um ou dois operandos e colocam o resultado em um terceiro operando.

Adição	Subtração	Multiplicação	Divisão	Negação
add[o][.]	subf[o][.]	mulhw[.]	divw[o][.]	neg[o][.]
addc[o][.]	subfc[o][.]	mulhwu[.]	divwu[o][.]	
adde[o][.]	subfe[o][.]	mulli		
addi	subfic	mullw[o][.]		
addic[.]	subfme[o][.]			
addis	subfze[o][.]			
addme[o][.]				
addze[o][.]				

Tabela 2.2: Intruções de Aritméticas

E	Não e	Ou	Não ou	Ou exclusivo	Equivalencia	Outros
and[.]	nand[.]	or[.]	nor[.]	xor[.]	eqv[.]	extsb[.]
andi.		ori		xori		extsh[.]
andis.		oris		xoris		cntlzw[.]
andc[.]		orc[.]				

Tabela 2.3: Intruções lógicas

Instruções de comparação comparam um registrador com outro registrador ou com um campo imediato (sufixo “i”) e colocam o resultado no registrador CR. Podem ser aritméticas (sinalizadas) ou lógicas (sufixo “l”, não sinalizadas).

Instruções de desvio podem ser de desvio incondicional ou condicional. As de desvio condicional testam algum bit de algum campo de registrador CR. Podem, opcionalmente, guardar o PC

Aritméticas	Lógicas
cmp	cmpl
cmpi	cmpli

Tabela 2.4: Intruções de comparação

atual no registrador LR (instruções com sufixo ”l”), podem também, opcionalmente, decrementar o valor contido no registrador CTR e testá-lo para determinar se o desvio acontecerá. O endereço alvo, pode ser um endereço relativo, absoluto (sufixo “a”) ou um endereço contido no registrador LR ou CTR.

Desvio
b[l][a]
bc[l][a]
bcctr[l]
bclr[l]

Tabela 2.5: Intruções de desvio

Instruções lógicas sobre o CR são instrução que permitem que seja feita uma operação lógica entre dois bits do CR e o resultado seja colocado em outro bit do CR.

Instruções lógicas sobre o CR	
crand	cror
crandc	crorc
creqv	crxor
crnand	mcrf
crnor	

Tabela 2.6: Intruções lógicas sobre o CR

Existem outras instruções como instruções que manipulam a cache, gerenciam interrupções dentre outras que não serão abordadas aqui pois não fazem parte do escopo deste trabalho. Essas instruções são privilegiadas, por isso o cross-compiler adotado para a geração do código dos *benchmarks* não irá gerá-las.

Rotação	Deslocamento	Deslocamento à direita algébrico
rlwinm[.]	slw[.]	sraw[.]
rlwnm[.]	srw[.]	srawi[.]
rlwimi[.]		

Tabela 2.7: Intruções de rotação e deslocamento

2.1.3 Ponto flutuante

Por ser voltado ao mercado de sistemas embutidos, esse processador não possui instruções de ponto flutuante, mas isso não significa que não se possa fazer um programa que lide com dados em ponto flutuantes. Isso é possível através da emulação de ponto flutuante, há dois modos de fazer isso. O método recomendável é através de sub-rotinas que façam essa emulação. Outro meio de contornar isso, parte do princípio de que quando uma instrução de ponto flutuante chega ao processador ela não vai ser reconhecida e uma interrupção de instrução ilegal será gerada. Com isso é possível que o tratador de interrupções determine qual instrução deveria ser decodificada e trate essa instrução com uma biblioteca de ponto-flutuante implementada com instruções inteiras. Esse modo não é o mais apropriado, pois a cada instrução de ponto flutuante é gerada uma interrupção que causa uma troca de contexto, entretanto com esse método é possível executar código gerado para processadores PowerPC que tenham instruções de ponto flutuante.

2.2 Modelo funcional

Um modelo em ArchC é dividido em 3 arquivos, para o modelo deste trabalho os arquivos são:

- powerpc.ac - Possui informação sobre os registradores, memória, ordem dos bytes e tamanho da palavra de dados.
- powerpc-isa.ac - Possui informação sobre os formatos de instrução, campos, instrução, op-codes e assembly.
- powerpc_isa.cpp - Possui o comportamento (descrito em SystemC) das instruções.

Após criados esses dois arquivos é necessário chamar o pré-processador ArchC, ele criará um conjuntos de classes em SystemC que futuramente serão o simulador. Mas antes

```
AC_ARCH(powerpc){
  ac_mem     MEM:5M;
  ac_regbank GPR:32;  // General-Purpose Registers
  ac_wordsize 32;

  ac_format Fmt_XER = "%so:1 %ov:1 %ca:1 %reserved:22 %bc:7";
  ac_reg CR;            // Condition Register
  ac_reg CTR;           // Count Register
  ac_reg LK;            // Link Register
  ac_reg<Fmt_XER> XER;  // XER Register

  ARCH_CTOR(powerpc) {
    ac_isa("powerpc_isa.ac");
    set_endian("big");
  };
};
```

Figura 2.11: Código do arquivo powerpc.ac

é necessário modificar o arquivo “powerpc_isa.cpp” que possui o comportamento (descrito em SystemC[4]) das instruções. Para cada instrução é criada uma função que conterá o comportamento da mesma, o mesmo ocorre para os formatos (caso seja necessário definir comportamentos comuns a instruções de um formato) e existe uma função que define o comportamento de todas as instruções. Funções para ler e escrever na memória e nos registradores são disponibilizadas pelo ArchC, assim como uma variável chamada ac_pc, que possui o *program counter* do simulador.

Para o modelo poder tirar todo o proveito de um cross-compiler, é necessário que ele tenha suporte a chamadas de sistema[7]. Para isso, é passado a flag “-abi” para o pré-processador ArchC e é necessário a criação de um arquivo chamado “powerpc_syscall.cpp” que contém as informações sobre a ABI (Appliction Binary Interface) do PowerPC. Métodos como get_int e set_int dentre outros são usados pelo ArchC para saber em quais registradores são passados os parâmetros para uma função. Informações sobre como ler e escrever buffers e como voltar de uma chamada de função também são necessárias. A ABI do PowerPC define que em chamadas de

```
ac_format Type_D   = "%op:6 %rt:5 %ra:5 %imm:16:s";
ac_format Type_XO  = "%op:6 %rt:5 %ra:5 %rb:5 %oe:1 %xo:9 %rc:1";
ac_format Type_X   = "%op:6 %rt:5 %ra:5 %rb:5 %eo:10 %rc:1";
ac_format Type_M   = "%op:6 %rt:5 %ra:5 %sh:5 %mb:5 %me:5 %rc:1";
ac_format Type_XFX = "%op:6 %rt:5 %spr:10 %eo:10 %rc:1";
ac_format Type_I   = "%op:6 %li:24:s %aa:1 %lk:1";
ac_format Type_B   = "%op:6 %bo:5 %bi:5 %bd:14:s %aa:1 %lk:1";
ac_format Type_XL  = "%op:6 %crfD:3 %na1:2 %crfS:3 %na2:2 %na3:5
                     %eo:10 %rc:1";
```

Figura 2.12: Trecho de código descrevendo os formatos de instruções no arquivo powerpc-isa.ac

```
ac_instr<Type_D> addi, addic, addic_dot, addis, mulli, subfic;
ac_instr<Type_D> andi_dot, andis_dot, ori, oris, xori, xoris;
ac_instr<Type_D> lbz, lbzu, lha, lhau, lhz, lhzu, lmw, lwzu, lwz;
ac_instr<Type_D> stb, stbu, sth, stwu, stmw, sthu, stw, cmpi, cmpli;
```

Figura 2.13: Trecho de código descrevendo quais instruções pertencem ao formato D no arquivo powerpc-isa.ac

funções, os registradores r3 até r10 são usados para passar parâmetros à funções e os registradores r3 e r4 são usados para retornar valores de funções.

2.3 Validação do modelo

A validação foi feita através de um conjunto de *benchmarks* disponíveis ao público, juntamente com código próprio. Para iniciar a validação foi escolhido um *benchmark* simples para facilitar a depuração dos eventuais erros no modelo. Com o decorrer dos testes os *benchmarks* foram ficando mais complexos.

O primeiro *benchmark* escolhido foi o do projeto Dalton [5], voltado para o microcontrolador i8051, por ser extremamente simples. No código fonte original os resultados estavam sendo escritos em um registrador específico do micro-controlador, para poder usar o mesmo

```
add.set_asm("add %rt, %ra, %rb");
add.set_decoder(op=0x1F, xo=0x10A,rc=0,oe=0);

add_dot.set_asm("add. %rt, %ra, %rb");
add_dot.set_decoder(op=0x1F, xo=0x10A,rc=1,oe=0);
```

Figura 2.14: Trecho de código descrevendo as instruções add e add. no arquivo powerpc-isa.ac

```
void ac_behavior( instruction ){
  ac_pc = ac_pc +4;
}
```

Tabela 2.8: Comportamento comum a todas as instruções é o de incrementar o PC

no modelo do PowerPC foi preciso mudar o destino dos resultados do registrador para a posição me memória 0x0 que foi monitorada para a comparação dos resultados. Os resultados podem ser encontrados na tabela 2.9.

Programa	Instruções executadas
negcnt.c	159
gcd.c	231
int2bin.c	167
cast.c	105
divmul.c	181
fib.c	491
sort.c	2.773
sqroot.c	2.742
xram.c	59.396

Tabela 2.9: Benchmark do projeto Dalton[5]

Após a primeira etapa de validação foram escritos 6 algoritmos de ordenação baseados nos disponíveis em [6]. Foram usadas como entradas 10000 números gerados aleatoria-

```
void ac_behavior( stw ) {
  unsigned valor = ra == 0 ? 0 : GPR.read(ra);
  MEM.write( valor+imm, GPR.read(rt));
}
```

Figura 2.15: Exemplo de comportamento de uma instrução

```
int powerpc_syscall::get_int(int argn) {
  return GPR.read(3+argn);
}
void powerpc_syscall::set_int(int argn, int val) {
  GPR.write(3+argn, val);
}
```

Figura 2.16: Métodos para ler parâmetros passados para uma função

mente antes da execução do *benchmark* que foram usados em todos os algoritmos. Os programas rodaram em uma maquina Pentium III 600Mhz, obtendo os resultados conforme a tabela 2.10.

Algorítmo	Instruções executadas	Velocidade em KI/s	Tempo de execução (s)
Bubble	20.252.506	131.75	181,96
Merge	684.938	130.71	5,24
Binary Insertion	11.427.188	132.53	86,22
Straight Insertion	14.132.519	131.65	107,35
Bucket	113.074	125.64	0,9

Tabela 2.10: Algoritmos de ordenação executados no simulador do PowerPC 405

Para comaparação de resultados, os mesmos algoritmos foram executados no modelo do MIPS, conforma a tabela 2.11 e no modelo do SPARC, conforme a tabela 2.12.

Os gráficos 2.17 e 2.18 mostram de maneira mais clara o comportamento dos modelos.

Algorítmo	Instruções executadas	Velocidade em KI/s	Tempo de execução (s)
Bubble	28.395.587	121.03	234,62
Merge	835.894	119.58	6,99
Binary Insertion	15.404.049	120.91	127,33
Straight Insertion	19.229.397	121.34	158,47
Bucket	122.066	117.37	1,04

Tabela 2.11: Algoritmos de ordenação executados no simulador do MIPS

Algorítmo	Instruções executadas	Velocidade em KI/s	Tempo de execução (s)
Bubble	20.510.914	112,72	181,96
Merge	644.858	112,31	5,92
Binary Insertion	11.216.631	112,53	99,68
Straight Insertion	13.878.630	112,66	123,19
Bucket	97.050	103,24	0,94

Tabela 2.12: Algoritmos de ordenação executados no simulador do SPARC

Para finalizar a validação, foram realizados os *benchmarks* MiBench[9], voltados para a o mercado embutido. Para essas simulações foi utilizado um Pentium IV 1.8Ghz

Algorítmo	Instruções executadas	Velocidade em KI/s	Tempo de execução (s)
Search string (small)	241.076	231,80	1,04
Search string (large)	6.755.910	262,37	25,75
Quick Sort (small)	14.996.260	294,97	50,84

Tabela 2.13: Algoritmos referentes ao *benchmark* MiBench executados sobre o simulador do PowerPC 405

Para comparação, os mesmos *benchmarks* foram executados com os modelos dos processadores MIPS (tabela 2.14) e Sparc (tabela 2.15), disponíveis na página do ArchC.

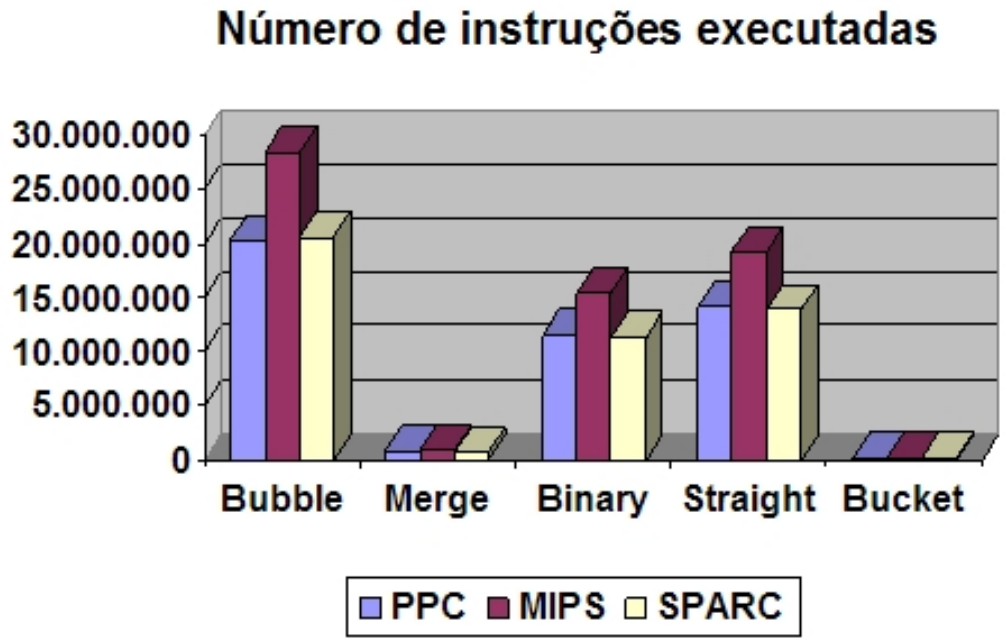

Figura 2.17: Comparação do número de instruções executadas pelos algoritmos de ordenação

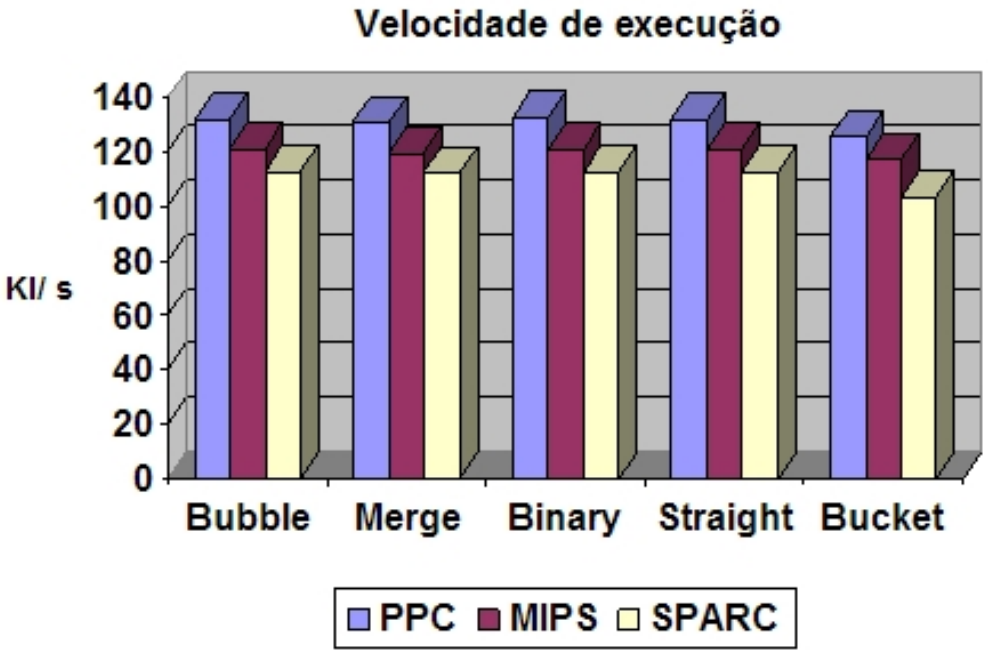

Figura 2.18: Comparação da velocidade de execução dos algoritmos de ordenação (milhares de instruções por segundo)

Algorítmo	Instruções executadas	Velocidade em KI/s	Tempo de execução
Search string (small)	279.725	259,00	1,08
Search string (large)	6.967.897	267,58	26,04
Quick Sort (small)	14.510.445	289,11	50,19

Tabela 2.14: Algoritmos referentes ao *benchmark* MiBench executados sobre o simulador do MIPS

Algorítmo	Instruções executadas	Velocidade em KI/s	Tempo de execução (s)
Search string (small)	268287	237.42	1,13
Search string (large)	6.645.569	237,68	27,96
Quick Sort (small)	14.217.285	255,16	55,72

Tabela 2.15: Algoritmos referentes ao *benchmark* MiBench executados sobre o simulador do SPARC

Capítulo 3

O Gerador de Montadores

Na tentativa de agilizar o projeto de sistemas embarcados, os projetistas buscam a automação de certos passos deste processo. Um destes passos, crucial na hora de se realizarem testes, é a criação de um compilador e um montador para o micro-processador presente no sistema. No caso de micro-processadores amplamente difundidos e utilizados em plataformas para sistemas embarcados, pode-se encontrar com relativa facilidade conjuntos de ferramentas que incluem compilador e montador. Mas em casos como o dos ASIPs grande parte destas ferramentas precisa ser criada, e é neste contexto que surge a idéia do programa gerador de montadores: uma ferramenta capaz de gerar automaticamente um montador a partir de uma descrição do micro-processador alvo feita utilizando uma ADL.

3.1 Descrição Funcional

A idéia básica do Gerador de Montadores é tomar uma descrição do processador feita em ArchC e a partir dela extrair todas as informações necessárias para a criação de um montador para o processador em questão. Partindo deste princípio, verifiquemos quais são as informações contidas na descrição ArchC:

- Os formatos de instrução existentes na arquitetura
- Os campos que fazem parte de cada formato, além do tamanho de cada campo e uma indicação se ele é um campo sinalizado ou não
- As instruções pertencentes a cada um dos formatos descritos

- O código *assembly* de cada uma das instruções e os valores dos campos para que a instrução seja decodificada (para os casos de campos que tem valor fixo, como o campo do código operacional da instrução, por exemplo)

Tendo estas informações, deve ser gerado automaticamente o montador, que seja capaz de ler um programa escrito na linguagem *assembly* do processador e gerar o código de máquina referente a este programa, pronto para ser rodado no simulador ou mesmo em uma plataforma FPGA.

No entanto, constatamos que algumas informações necessárias para a geração do montador não estão presentes na descrição ArchC, são elas:

- Quais instruções são de desvio (absoluto ou relativo) e qual o campo da instrução contém o endereço alvo do desvio (informação necessária para o tratamento de *labels* alfanuméricos no código *assembly* gerado pelo compilador)

- Como tratar as chamadas pseudo-instruções, que são instruções que não fazem parte da arquitetura, mas são geradas pelo compilador e depois mapeadas para instruções nativas

- Como deve ser calculado o endereço alvo efetivo das instruções de desvio (por exemplo, tomar o valor do endereço alvo e deslocar dois bits para a esquerda)

Como veremos na seção a seguir, estas informações devem ser passadas ao gerador de montadores de alguma forma, seja incluindo elas na descrição ArchC do processador ou em algum arquivo separado, para que o arquivo contendo o código *assembly* gerado pelo compilador seja pré-processado antes de passar pelo montador (gerado automaticamente).

3.2 Implementação

O Gerador de Montadores foi implementado na linguagem C++. Ele é um programa que funciona como um *parser* da descrição ArchC do processador, o qual extrai da descrição as informações necessárias para a tradução de um código *assembly* em código de máquina. Desta forma, definimos uma gramática livre de contexto (GLC) no formato BNF que representa a estrutura da descrição ArchC e utilizamos o Gerador de Analisadores Léxicos e Sintáticos (GALS), um software desenvolvido na UFSC como trabalho de conclusão de curso [1], para gerar um *parser* descendente do tipo LL(1). A partir de uma gramática livre de contexto (GLC), além de uma

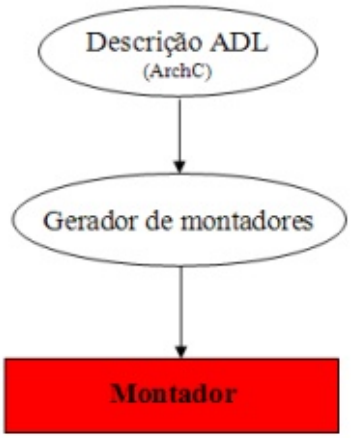

Figura 3.1: Fluxo para geração do montador

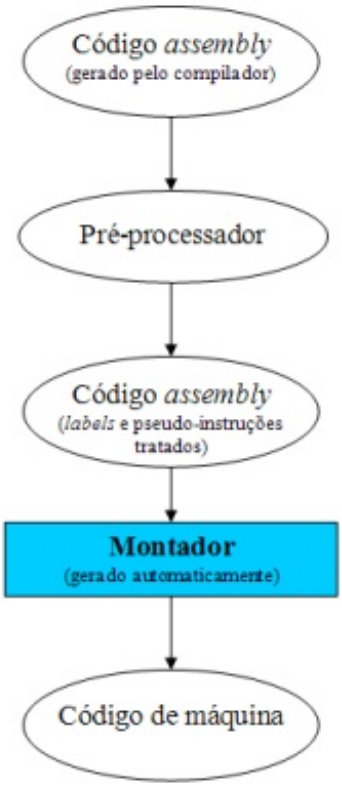

Figura 3.2: Fluxo para geração de código de máquina

lista de tokens e definições regulares, o GALS gera o analisador léxico e o analisador sintático automaticamente enquanto que o analisador semântico deve ser implementado manualmente.

Como vemos na figura 3.3, as informações sobre a arquitetura do conjunto de instruções são todas extraídas da descrição ArchC (pelo analisador semântico), e com elas é gerada

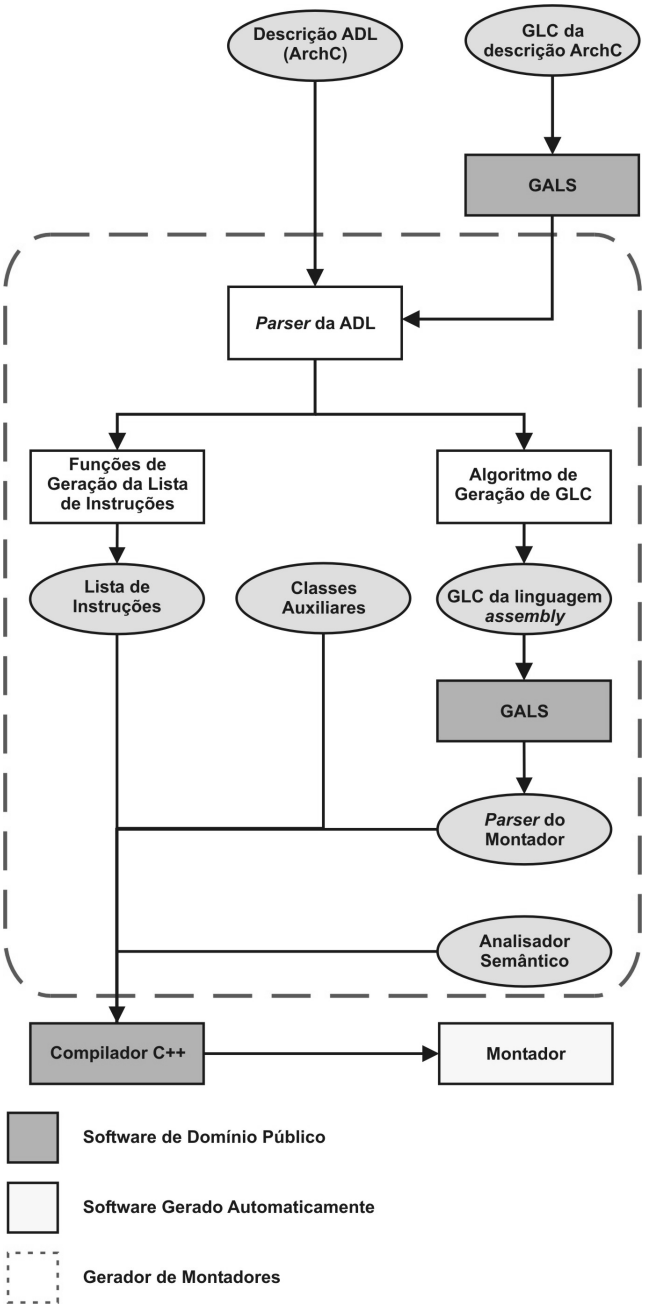

Figura 3.3: Fluxo detalhado para geração automática do montador

uma lista com todas as instruções além de informações como os campos de cada instrução, quais campos tem valor fixo, quais são sinalizados, quais instruções são de desvio, etc.

Além desta lista, é gerada uma GLC no padrão BNF que representa a estrutura do código *assembly* do processador descrito. Como o GALS é um programa de código aberto, surge assim a possibilidade de utilizarmos este mesmo software para gerar um *parser* da linguagem *assembly* do processador a partir da GLC gerada, o qual seria a base do montador.

Compilando juntos o *parser*, a lista de instruções e um analisador semântico do código *assembly* (que é sempre o mesmo para qualquer arquitetura), temos pronto o montador.

Como foi dito na seção anterior, inicialmente as informações presentes na descrição ArchC pareceram suficientes para a geração do montador, mas verificamos que não havia nada na descrição ArchC que identificasse quais instruções eram de desvio. Esta informação é necessária, pois o montador pode encontrar um *label* alfanumérico como sendo o valor de um dos campos de uma instrução, e neste caso é necessário que ele saiba que aquela instrução é de desvio e qual dos campos da instrução deve armazenar o valor do endereço alvo deste desvio. Decidimos então que era necessário adicionar esta informação de alguma forma na descrição do processador, e por isso sugerimos uma extensão da linguagem ArchC, adicionando duas novas diretivas para indicar quais instruções são desvio (absoluto e relativo) e qual é o campo do formato da instrução que deverá conter o endereço alvo.

Para indicar que uma instrução é de desvio absoluto, utilizamos a palavra reservada ac_addr_mode_A seguida do nome da instrução e do nome do campo que armazena o endereço alvo do desvio. Utilizando a palavra reservada ac_addr_mode_R desta mesma maneira, indicamos que uma instrução é de desvio relativo.

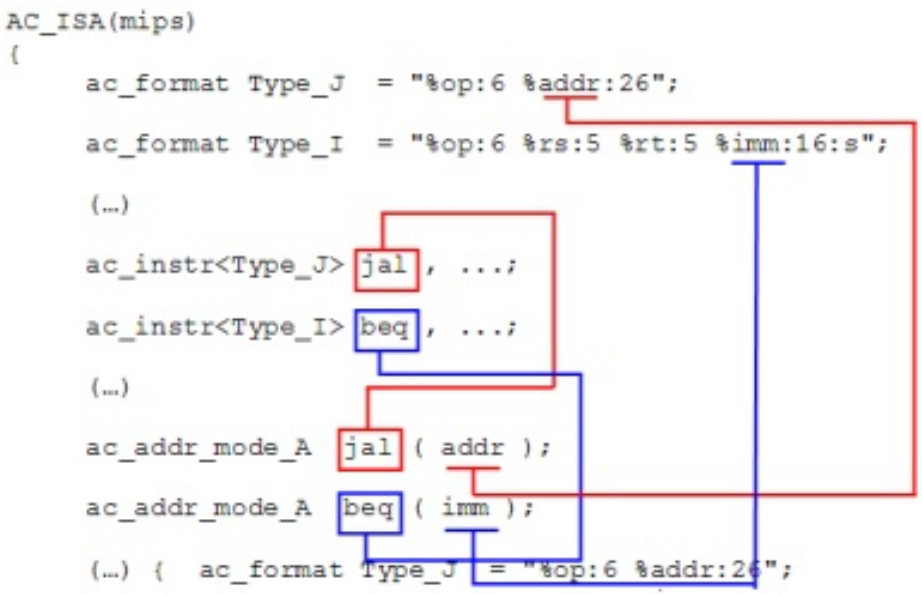

Figura 3.4: Descrição ArchC com as diretivas para indicar quais instruções são de desvio

Mas mesmo com esta nova informação, dificuldades surgem quando envolvemos as pseudo-instruções. Justamente por não fazerem parte da arquitetura do conjunto de instruções, estas pseudo-instruções não devem aparecer na descrição ArchC, o que nos levou à necessidade de

incluir estas informações em um arquivo separado, o qual seria lido por um pré-processador.

Foi definido então um padrão para se fazer o mapeamento de uma pseudo-instrução em uma ou mais instruções nativas. O pré-processador toma então como entrada o arquivo *assembly* que é gerado pelo compilador e este arquivo com o mapeamento das pseudo-instruções, e gera um novo arquivo *assembly* contendo apenas instruções nativas. Como vemos na figura 3.4, uma pseudo-instrução pode ser mapeada em uma ou mais instruções nativas. Para mapear uma pseudo-instrução para apenas uma instrução nativa, basta escrever o código *assembly* da pseudo-instrução (com os nomes dos campos sendo iguais aos nomes definidos na descrição ArchC) seguida do código *assembly* da instrução nativa correspondente (também com os nomes dos campos sendo iguais àqueles definidos na descrição ArchC). Quando uma pseudo-instrução deve ser mapeada para mais de uma instrução nativa, deve ser usada a diretiva AC_MACRO, seguida do código *assembly* da pseudo-instrução e dos códigos *assembly* das instruções nativas correspondentes. No exemplo da figura 3.5, a pseudo-instrução “cmpwi”, que contém três campos, deve ser mapeada para a instrução nativa “cmpi”, que contém estes mesmo três campos e mais um campo adicional (o campo “li”), o qual tem seu valor fixado em zero. Ainda na mesma figura, a pseudo-instrução “xyz”, que contém dois campos, deve ser mapeada para duas instruções nativas (“add” e “lbz”), as quais usam estes dois campos e mais alguns campos adicionais.

```
AC_PRE_PROCESSOR {
AC_PSEUDO_INSTRUCTIONS {
cmpwi %cf, %ra, %imm :: cmpi %cf, %li=0, %ra, %imm;
    AC_MACRO xyz %e, %f ::
    {
      add %e, %b=2, %w=1;
      lbz %f, %b=0(%c=0xFF);
    }
};
```

Figura 3.5: Mapeamento de pseudo-instruções em instruções nativas

Além deste mapeamento de pseudo-instruções em instruções nativas, outra característica que também varia de uma arquitetura para outra e que não está presente na descrição ArchC é a maneira com que o endereço alvo efetivo de uma instrução de desvio é calculado. Primeiramente, é preciso substituir todos os *labels* alfanuméricos por valores que representem o

endereço alvo (levando-se em conta se a instrução é de desvio absoluto ou relativo), mas em muitas ocasiões este não é o endereço alvo efetivo e é preciso fazer algum tipo de operação sobre ele antes de ser gerado o código de máquina da instrução (deslocar o valor dois bits para a esquerda, somar dois bytes no valor, e assim por diante). Isto também poderia ser incluído no arquivo de configuração e tratado pelo pré-processador, com o usuário definindo, para cada instrução de desvio, as operações que devem ser feitas sobre o campo do endereço alvo. Para não precisar definir um padrão de quais símbolos o usuário pode usar para compor esta operação que será feita sobre o campo do endereço, decidimos que deveria ser gerada uma classe contendo métodos que calculam o endereço efetivo de um desvio (um método para cada instrução de desvio), sendo que o corpo do método seria a operação definida no arquivo de configuração. Desta forma, a operação deveria ser validada pelo compilador C++, ou seja, seria necessário compilar o pré-processador antes de utiliza-lo para a tradução dos *labels* alfanuméricos nos endereços alvo efetivos das instruções de desvio. Surge assim a oportunidade da criação de um gerador de pré-processadores, o qual lê as informações do arquivo de configuração e a partir delas gera um pré-processador para a arquitetura desejada, o qual é capaz de substituir pseudo-instruções por instruções nativas e substituir todos os *labels* alfanuméricos por valores que representam o endereço alvo efetivo de uma instrução de desvio (o qual é calculado de acordo com a operação definida pelo projetista para aquela instrução).

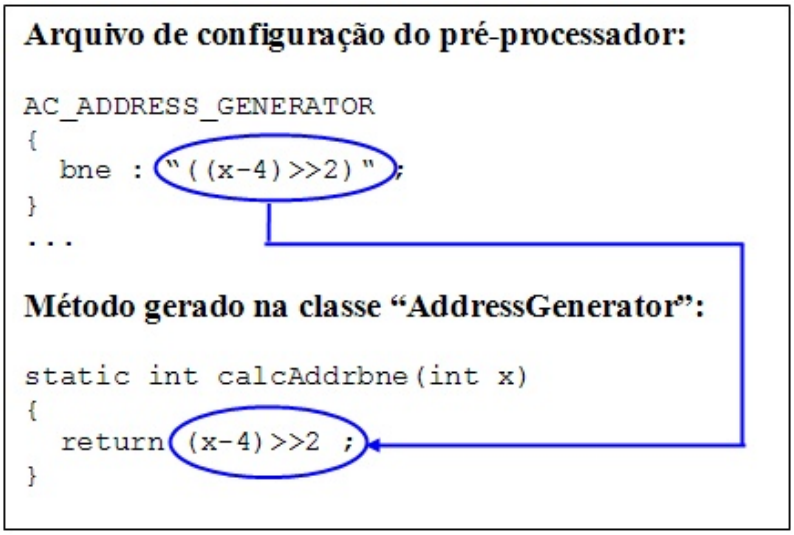

Figura 3.6: Definição de como é calculado o endereço alvo de um desvio

No exemplo da figura 3.6, foi definido que para a instrução *bne*, deve-se tomar o endereço que está definido no código *assembly* (representado por *x* no arquivo de configuração do pré-processador), subtrair dele quatro unidades e desloca-lo dois bits para a direita. Esta operação

torna-se o valor de retorno de um método da classe *AddressGenerator*, o qual será chamado pelo analisador semântico do pré-processador para o cálculo do endereço-alvo efetivo do desvio.

Seguindo a mesma idéia de tratar informações específicas de cada arquitetura, decidimos incluir a possibilidade de serem definidas no arquivo de configuração do gerador de pré-processadores palavras reservadas da linguagem *assembly* do processador, as quais devem ser mapeadas para valores numéricos. Por exemplo, em um código *assembly* do microprocessador MIPS provavelmente encontraremos “$sp” como sendo um dos campos de várias instruções. Neste caso, “$sp” se refere ao valor do apontador da pilha (*stack pointer*), que no processador fica armazenado no registrador 29, ou seja, toda referência a “$sp” deve ser trocada pelo valor 29 (ou 0x1D). Além disso, é possível incluir no arquivo de configuração do pré-processador qual é o símbolo usado para inserir comentários no código *assembly* e quais símbolos devem ser ignorados, como por exemplo o símbolo “$”, que pode aparecer no código precedendo o nome ou número de um registrador.

```
AC_RESERVED_WORDS
{
  COMMENT_SYMBOL="#";
  IGNORE_SYMBOLS="$";

  sp=0x1D;
  fp=0x1E;
}
```

Figura 3.7: Definição de palavras reservadas, símbolo de comentário e símbolos a serem ignorados

Após passar pelo pré-processador, o código *assembly* está pronto para ser submetido ao montador, o qual vai gerar o código de máquina correspondente, pronto para ser rodado no simulador ou eventualmente em uma plataforma FPGA.

3.3 Experimentos

Para a validação do gerador de montadores até para uma avaliação de desempenho, podemos compilar alguns programas em C++, tomar o código *assembly* gerado e submete-lo

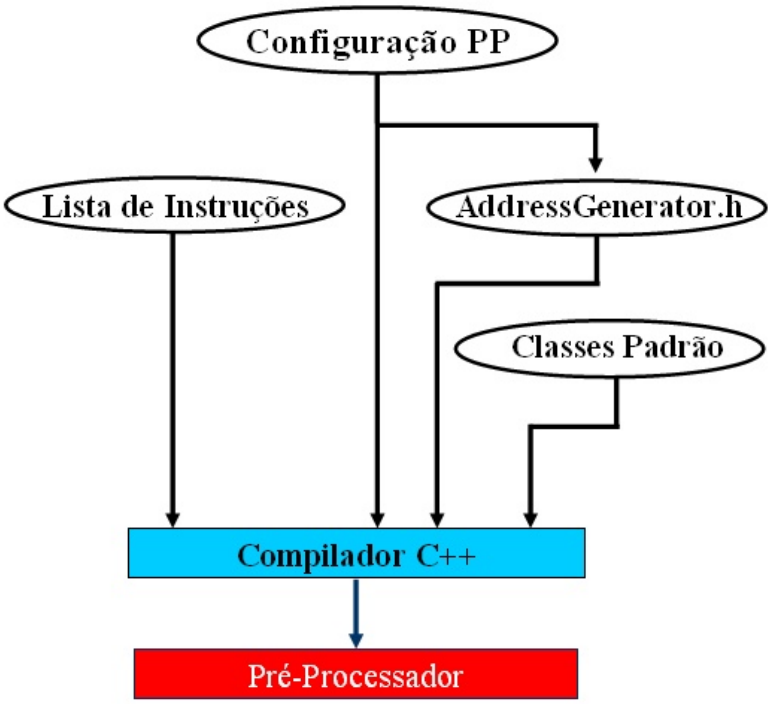

Figura 3.8: Diagrama do fluxo de geração do pré-processador

ao pré-processador e ao montador gerados e em seguida rodar o código de máquina correspondente no simulador SystemC gerado pelo ArchC a partir da descrição do processador.

3.3.1 Configuração dos experimentos

Os programas adotados para validar os geradores foram extraídos do conjunto de *benchmarks* do Projeto Dalton [5]. A fim de gerar códigos *assembly* para testar os montadores gerados automaticamente, utilizaram-se os compiladores cruzados gcc (para o MIPS e o PowerPC) e CCS (para o PIC). Os tempos de execução medidos utilizaram o comando *time* do Linux, somando-se os tempos de CPU denominados *user* e *system*.

Os experimentos aqui relatados foram realizados em um computador PC com CPU Pentium 4, operando à freqüência de 1.8 GHz, com 256 MB de memória principal, sob o sistema operacional Linux Debian, com *kernel* versão 2.4.25-1-686.

3.3.2 Procedimentos experimentais

Para cada CPU-alvo Ci, representada através de um modelo Mi descrito em ArchC, foram geradas as seguintes ferramentas:

- O simulador do conjunto de instruções Si (usando o pacote ArchC);
- O pré-processador Pi;
- O montador Ai;

Posteriormente, cada programa *benchmark* foi compilado para cada CPU-alvo Ci, gerando os respectivos códigos *assembly*. Por sua vez, os códigos *assembly* de cada *benchmark* foram submetidos ao pré-processador Pi e, em seguida, ao montador Ai, resultando no código binário para a CPU-alvo Ci.

3.3.3 Resultados experimentais

Os resultados apresentados a seguir foram obtidos através de trabalho cooperativo entre os autores e o aluno de mestrado Leonardo Taglietti e estão resumidos em artigo cientifico onde o referido aluno é co-autor [11].

A tabela 3.1 mostra o número de instruções executadas pelas CPUs-alvo para cada programa *benchmark*. O acrônimo N/A (não aplicável) indica casos não testados. Por exemplo, o programa *divmul* não pôde ser testado no MIPS, pois necessita da instrução *break*, que não foi implementada no modelo disponível em [3]. O programa *xram* não pôde ser testado no PIC, devido a restrições de memória. Note que para o PowerPC o número de instruções é ligeiramente inferior às mostradas na tabela 2.9, pois, para ter uma simulação mais precisa, os resultados não estão sendo mais guardados na posição de memória 0x0, resultando assim em um número menor de instruções a executar.

Os tempos necessários para gerar cada um dos pré-processadores e montadores são mostrados nas Tabelas 3.2 e 3.3, respectivamente. Nestas tabelas, a segunda coluna representa o número de instruções descritas no modelo ArchC.

Note que os tempos de geração do pré-processador (Tabela 3.2) são de mesma ordem de magnitude, embora o tempo do PIC seja cerca de 50% menor devido ao menor número de instruções desta CPU.

Observe que o tempo de geração do montador (Tabela 3.3) para o PowerPC 405 é cerca de três vezes maior que o tempo das demais CPUs. Este tempo está correlatado ao maior número de instruções do PowerPC 405. A Tabela 3.4 mostra o número de instruções montadas para cada programa *benchmark*, para cada CPU-alvo.

Benchmark	MIPS	PowerPC	PIC
negcnt	154	121	150
gcd	209	198	125
int2bin	153	143	256
cast	80	87	68
divmul	N/A	167	296
fib	490	445	358
sort	2982	2744	2227
xram	43013	36869	N/A

Tabela 3.1: Número de instruções executadas

CPU	Instruções descritas	Tempo [s]
MIPS	58	0,04
PowerPC	120	0,05
PIC	35	0,025

Tabela 3.2: Tempos de geração do pré-processador

CPU	Instruções descritas	Tempo [s]
MIPS	58	1,13
PowerPC	120	3,44
PIC	35	1,06

Tabela 3.3: Tempos de geração do montador

Os tempos de pré-processamento e montagem de cada *benchmark* são mostrados na Tabela 3.5.

A correlação dos dados das Tabelas 5 e 6 mostra que, para uma mesma CPU, o tempo cresce em menor proporção do que o número de instruções montadas. Por exemplo, para o MIPS, quando se passa do programa *negnct* para o programa *sort*, o número de instruções é multiplicado por 5, enquanto o tempo é multiplicado por 2,3. Este comportamento mostra a eficiência das técnicas aqui propostas. Por outro lado, para um mesmo programa *benchmark* testado em CPUs distintas, o tempo pode crescer em proporção maior do que o número de instruções

Benchmark	**Instruções montadas**		
	MIPS	PowerPC	PIC
negcnt	32	28	30
gcd	53	48	32
int2bin	36	35	36
cast	48	55	60
divmul	N/A	55	106
fib	106	96	70
sort	154	148	110
xram	42	38	N/A

Tabela 3.4: Número de instruções montadas

Benchmark	**Tempo [s]**		
	MIPS	PowerPC	PIC
negcnt	0.030	0.035	0.020
gcd	0.030	0.050	0.020
int2bin	0.040	0.040	0.030
cast	0.040	0.050	0.030
divmul	N/A	0.050	0.060
fib	0.060	0.080	0.040
sort	0.070	0.110	0.050
xram	0.040	0.045	N/A

Tabela 3.5: Tempos de pré-processamento e montagem de cada programa

montadas. No programa *sort*, por exemplo, quando se passa do PIC para o PowerPC, o número de instruções é multiplicado por 1,3, enquanto o tempo é multiplicado por 2,2. Isto deve-se à o modelo do PowerPC ter mais de 3 vezes o número de instruções do PIC.

Capítulo 4

Conclusão

4.1 Contribuições

Durante o desenvolvimento deste trabalho, a linguagem de descrição de arquiteturas ArchC mostrou-se de grande utilidade no processo do desenvolvimento de software para sistemas embarcados. Ela permite ao desenvolvedor descrever diversas características diferentes de um processador para em seguida gerar um simulador com o qual é possível testar o software antes mesmo de se ter o hardware disponível fisicamente.

No entanto, quando da tentativa de se automatizar o processo de criação de um montador para o processador modelado partindo da sua descrição feita em ArchC, notou-se que algumas informações necessárias não estão presentes na descrição da arquitetura do conjunto de instruções feita nesta ADL. Isso abriu caminho para que se propusesse uma extensão da linguagem, com novas diretivas que permitem ao desenvolvedor definir quais das instruções da arquitetura são instruções de desvio. Desta forma, o ArchC tem seu poder de descrição aumentado e ainda facilita na geração automática de montadores.

4.2 Produtos de Trabalho

Com a extensão do ArchC, foi possível criar um programa capaz de tomar uma descrição funcional de um processador feita com esta ADL e extrair dela todas as informações necessárias para se gerar de forma automática um montador para a linguagem *assembly* do processador modelado.

Entretanto, há características que variam de uma arquitetura para outra, como

as pseudo-instruções e a maneira de se fazer o cálculo do endereço efetivo de uma instrução de desvio, características que optamos por não serem descritas junto com a arquitetura do conjunto de instruções (sendo mais uma extensão do ArchC) já que a principal função do montador é apenas gerar o código objeto a partir do código *assembly*, sem precisar saber como resolver os endereços de instruções de desvio ou como mapear as pseudo-instruções para instruções nativas. Estas características devem ser descritas em um arquivo de configuração separado, o qual é submetido a um novo programa que gera automaticamente um pré-processador para a arquitetura. Este pré-processador complementa o montador pois recebe como entrada um arquivo com o código *assembly* gerado pelo compilador (com *labels* alfanuméricos, pseudo-instruções, comentários, etc.) e submete ao montador um novo arquivo com o código *assembly* pronto para ser transformado em código objeto.

Para efeitos de validação e estudo de caso, foi escolhido o processador PowerPC 405, bastante utilizado em sistemas embarcados. Desta forma, foi criado um modelo funcional em ArchC deste processador e sua validação foi feita com diferentes *benchmarks* e algoritmos de ordenação, e o mesmo modelo foi aproveitado no processo de validação do gerador de montadores e do gerador de pré-processadores.

4.3 Dificuldades encontradas

Como foi dito na seção 3.3, foram gerados com sucesso montadores para diferentes arquiteturas de forma automática. Mas algumas dificuldades foram encontradas tanto no processo de desenvolvimento do gerador de montadores e do gerador de pré-processadores quanto no processo de validação dos dois programas.

O montador que é gerado automaticamente funciona como um *parser* da linguagem *assembly*, desta forma o gerador de montadores gera uma GLC que é utilizado pelo GALS para criar este *parser*. Como foi explicado na seção 3.2, cada instrução definida na descrição funcional do processador torna-se uma produção desta GLC, mas quando o número de produções da gramática ultrapassa um certo número (que aparentemente é algo em torno de 100 a 110 produções), o GALS gera um *parser* que não funciona. Este parece ser um problema apenas na geração das classes C++ que funcionam como *parser* porque, utilizando-se a mesma gramática, o simulador do GALS funciona perfeitamente no reconhecimento de um mesmo código que o *parser* gerado não consegue reconhecer. Esta dificuldade foi encontrada no momento da validação do montador gerado para o PowerPC 405. Como este processador possui cerca de 150 instruções, foi

preciso excluir do modelo funcional aquelas que não eram utilizadas pelos *benchmarks* que seriam utilizados na validação.

Outra dificuldade, que também ocorreu durante a validação, foi que o ArchC parece não conseguir decodificar corretamente arquivos com código hexadecimal muito extensos. O problema normalmente ocorria nos algoritmos de ordenação: na primeira parte do programa era feito o carregamento dos valores a serem ordenados. Nesta parte, uma rotina de 5 instruções é executada diversas vezes (uma vez para cada valor), mas no meio de uma destas rotinas o ArchC não conseguia decodificar a instrução (normalmente uma mesma instrução que já havia sido decodificada diversas vezes antes no mesmo código).

Além disso, o que dificultou a geração automática do montador foi a falta de padronização do código que é gerado pelo GCC para as diferentes arquiteturas. Um exemplo disso é que para o PowerPC 405 o GCC gera diversas pseudo-instruções, as quais podem ser mapeadas para instruções nativas no arquivo de configuração do pré-processador pois todas elas têm mnemônicos diferentes de instruções nativas. Já no caso do MIPS, o GCC gera pseudo-instruções que têm mnemônicos iguais aos de instruções nativas, sendo que a diferença está apenas nos operandos da instrução. Por exemplo, a instrução ”jr”faz um desvio para o endereço armazenado em um registrador, sendo que o número deste registrador é o único operando da instrução. A instrução ”j”faz um desvio para um endereço absoluto, neste caso o único operando da instrução é o próprio endereço. Mas o GCC gera em alguns lugares do código uma instrução ”j”que tem como operando o número de um registrador. Desta forma não há como mapear esta pseudo-instrução para alguma instrução nativa, já que segundo o modelo funcional as instruções ”j”e ”jr”já são instruções nativas. Isto exigiu que os mnemônicos das instruções fossem alterados manualmente no código *assembly* gerado pelo GCC antes que este código fosse submetido ao pré-processador.

4.4 Trabalhos futuros

Tanto o gerador de montadores como o gerador de pré-processadores são ferramentas que podem ajudar muito no processo de desenvolvimento de software para sistemas embarcados. Algumas melhorias podem ser feitas, as quais podem aumentar as funcionalidades do montador que é gerado automaticamente, além de prevenir que sejam necessárias intervenções manuais no código *assembly*.

Os montadores gerados atualmente pelo gerador de montadores não possuem suporte à ligação de código, algo bastante comum principalmente em programas que utilizam

funções de bibliotecas padrão ou de bibliotecas definidas pelo próprio desenvolvedor. Isto diminui o número de aplicações que, após passarem por um compilador como o GCC, podem ser rodadas no modelo funcional do processador utilizando-se o montador que é gerado automaticamente.

Voltando ao caso das pseudo-instruções, há algumas em que um dos campos da instrução nativa correspondente é uma expressão aritmética feita sobre o valor de um dos operandos da pseudo-instrução. Neste caso não há como se fazer o mapeamento no arquivo de configuração do pré-processador, já que atualmente neste arquivo pode-se apenas mapear diretamente o valor de um campo da pseudo-instrução para um campo da instrução nativa, e isto pode ser outra melhoria a ser feita.

Uma terceira melhoria seria que o montador gerado automaticamente pudesse gerar código objeto no formato ELF (*Executable and Linking Format*, ou Formato Executável e Ligável). Os montadores atualmente gerados só geram o código objeto no formato hexadecimal do ArchC. Isso tornaria possível a execução do código não só no simulador ArchC, mas também em um processador real.

Referências Bibliográficas

[1] GUESSER, C. E. **GALS - Gerador de Analisadores Léxicos e Sintáticos**, TCC - Curso de Ciências da Computação, UFSC, Florianópolis, SC, 2002

[2] MARWEDEL, P. **Embedded System Design**. University of Dortmund.Kluwer Academic Publishers, Nov 2003.

[3] LABORATÓRIO DE SISTEMAS DE COMPUTAÇÃO DA UNICAMP. **The ArchC Architectural Description Language**. Disponível em http://www.archc.org

[4] SystemC Community. Disponível em http://www.systemc.org

[5] Dalton Project. Disponível em http://www.cs.ucr.edu/ dalton/i8051/i8051syn/

[6] Bruno R. Preiss. **Data Structures and Algorithms with Object-Oriented Design Patterns in Java**.

[7] Marcus Bartholomeu, Sandro Rigo, Rodolfo Azevedo, Guido Araujo. **Emulating Operating System Calls in Retargetable ISA Simulators.** IC- Technical Report 29 - December 2003.

[8] IBM Corporation, **PowerPC 405GP Embedded Processor User's Manual.** 11a. ed., 2003.

[9] Matthew R. Guthaus, Jeffrey S. Ringenberg, Dan Ernst, Todd M. Austin, Trevor Mudge, Richard B. Brown. **MiBench: A free, commercially representative embedded benchmark suite**. *IEEE 4th Annual Workshop on Workload Characterization*, Austin, TX, December 2001. Disponível em http://www.eecs.umich.edu/mibench/

[10] SBCCI-2003 - Tutorial sobre SystemC: Grant Martin

[11] Leonardo Taglietti, José O. Carlomagno Filho, Daniel C. Casarotto, Luiz C. Villar dos Santos, Olinto J. V. Furtado. **Geração Automática de Ferramentas para ASIPs a Partir de Linguagem de Descrição de Arquiteturas**, Artigo submetido ao *XI-IBERCHIP Workshop 2005*.

Capítulo 5

Anexos

5.1 Anexo I - Fontes do modelo do PowerPC 405

5.1.1 Arquivo powerpc.ac

```
/**************************************************
 *   Arquivo de descrição da arquitetura do PowerPC    *
 *   Autores:                                          *
 *   Daniel Carlos Casarotto                           *
 *   José Otávio Carlomagno Filho                      *
 **************************************************/


AC_ARCH(powerpc){

  ac_mem   MEM:5M;
  ac_regbank GPR:32;  // General-Purpose Registers

  ac_format Fmt_XER = "%so:1 %ov:1 %ca:1 %reserved:22 %bc:7";

  ac_reg CR;            // Condition Register
  ac_reg CTR;           // Count Register
  ac_reg LK;            // Link Register
  ac_reg<Fmt_XER> XER;  // XER Register

  ac_wordsize 32;

  ARCH_CTOR(powerpc) {
    ac_isa("powerpc_isa.ac");
    set_endian("big");
  };
};
```

5.1.2 Arquivo powerpc_isa.ac

```
/**************************************************
 *   Arquivo de descrição de instruções do PowerPC     *
 *   Autores:                                          *
 *   Daniel Carlos Casarotto                           *
 *   José Otávio Carlomagno Filho                      *
 **************************************************/


AC_ISA(powerpc){

  ac_format Type_D   = "%op:6 %rt:5 %ra:5 %imm:16:s";
  ac_format Type_XO  = "%op:6 %rt:5 %ra:5 %rb:5 %oe:1 %xo:9 %rc:1";
  ac_format Type_X   = "%op:6 %rt:5 %ra:5 %rb:5 %eo:10 %rc:1";
  ac_format Type_M   = "%op:6 %rt:5 %ra:5 %rb:5 %mb:5 %me:5 %rc:1";
  ac_format Type_XFX = "%op:6 %rt:5 %spr:10 %eo:10 %rc:1";
  ac_format Type_B   = "%op:6 %rt:5 %ra:5 %bd:14:s %aa:1 %rc:1";
  ac_format Type_I   = "%op:6 %li:24:s %aa:1 %rc:1";
  ac_format Type_XL  = "%op:6 %crfD:3 %na1:2 %crfS:3 %na2:2 %rb:5 %eo:10 %rc:1";


  ac_instr<Type_D>  addi,addic, addic_dot, addis, mulli, subfic;
  ac_instr<Type_D>  andi_dot, andis_dot, ori, oris, xori, xoris;
  ac_instr<Type_D>  lbz, lbzu, lha, lhau, lhz, lhzu, lmw, lwzu, lwz;
  ac_instr<Type_D>  stb, stbu, sth, stwu, stmw, sthu, stw, cmpi, cmpli;

  ac_instr<Type_XO> add, add_dot, addo, addo_dot;
  ac_instr<Type_XO> addc, addc_dot, addco, addco_dot, adde, adde_dot, addeo, addeo_dot;
  ac_instr<Type_XO> addme, addme_dot, addmeo, addmeo_dot, addze,
  ac_instr<Type_XO> addze_dot, addzeo, addzeo_dot;
  ac_instr<Type_XO> instr_neg, nego, neg_dot, nego_dot;
  ac_instr<Type_XO> subf, subf_dot, subfo, subfo_dot, subfc,
  ac_instr<Type_XO> subfc_dot, subfco, subfco_dot;
  ac_instr<Type_XO> subfe, subfe_dot, subfeo, subfeo_dot;
  ac_instr<Type_XO> subfme, subfme_dot, subfmeo, subfmeo_dot, subfze,
  ac_instr<Type_XO> subfze_dot, subfzeo, subfzeo_dot;
  ac_instr<Type_XO> divw, divw_dot, divwo, divwo_dot, divwu,
  ac_instr<Type_XO> divwu_dot, divwuo, divwuo_dot;
  ac_instr<Type_XO> mulhw, mulhw_dot, mulhwu, mulhwu_dot, mullw,
  ac_instr<Type_XO> mullw_dot, mullwo, mullwo_dot;

  ac_instr<Type_X>  instr_and, and_dot, andc, andc_dot, instr_or, or_dot;
  ac_instr<Type_X>  instr_nand, nand_dot, instr_nor, nor_dot, instr_xor, xor_dot;
  ac_instr<Type_X>  eqv, eqv_dot, orc, orc_dot;
  ac_instr<Type_X>  crand, crandc, creqv, crnand, crnor, cror, crorc, crxor;
  ac_instr<Type_X>  lbzux, lbzx, lhaux, lhax, lhbrx, lhzux, lhzx;
  ac_instr<Type_X>  lswi, lswx, lwarx, lwbrx, lwzux, lwzx;
  ac_instr<Type_X>  sthbrx, sthux, sthx, stswi, stswx;
  ac_instr<Type_X>  stwbrx, stwux, stwx, stbux, stbx;
```

```
ac_instr<Type_X>  bcctr, bcctrl, bclr, bclrl, cmp, cmpl;
ac_instr<Type_X>  extsb, extsb_dot, extsh, extsh_dot, cntlzw, cntlzw_dot;
ac_instr<Type_X>  slw, slw_dot, sraw, sraw_dot, srawi, srawi_dot, srw, srw_dot;
ac_instr<Type_X>  mfcr, mfmsr, mtmsr, rfi, sc, stwcx_dot;

ac_instr<Type_I>   b, ba, bl, bla;
ac_instr<Type_B>   bc, bca, bcl, bcla;
ac_instr<Type_M>   rlwimi, rlwimi_dot, rlwinm, rlwinm_dot, rlwnm, rlwnm_dot;
ac_instr<Type_XL>  mcrf, mcrxr;
ac_instr<Type_XFX> mfspr, mftb, mtspr, mtcrf;

ISA_CTOR(powerpc){

  // Addx
  add.set_asm("add %rt, %ra, %rb");
  add.set_decoder(op=0x1F, xo=0x10A,rc=0,oe=0);

  add_dot.set_asm("add. %rt, %ra, %rb");
  add_dot.set_decoder(op=0x1F, xo=0x10A,rc=1,oe=0);

  addo.set_asm("addo %rt, %ra, %rb");
  addo.set_decoder(op=0x1F, xo=0x10A,rc=0,oe=1);

  addo_dot.set_asm("addo. %rt, %ra, %rb");
  addo_dot.set_decoder(op=0x1F, xo=0x10A,rc=1,oe=1);

  // Addcx
  addc.set_asm("addc %rt, %ra, %rb");
  addc.set_decoder(op=0x1F, xo=0xA,rc=0,oe=0);

  addc_dot.set_asm("addc. %rt, %ra, %rb");
  addc_dot.set_decoder(op=0x1F, xo=0xA,rc=1,oe=0);

  addco.set_asm("addco %rt, %ra, %rb");
  addco.set_decoder(op=0x1F, xo=0xA,rc=0,oe=1);

  addco_dot.set_asm("addco. %rt, %ra, %rb");
  addco_dot.set_decoder(op=0x1F, xo=0xA,rc=1,oe=1);

  // Addex
  adde.set_asm("adde %rt, %ra, %rb");
  adde.set_decoder(op=0x1F, xo=0x8A,rc=0,oe=0);

  adde_dot.set_asm("adde. %rt, %ra, %rb");
  adde_dot.set_decoder(op=0x1F, xo=0x8A,rc=1,oe=0);

  addeo.set_asm("addeo %rt, %ra, %rb");
  addeo.set_decoder(op=0x1F, xo=0x8A,rc=0,oe=1);
```

```
addeo_dot.set_asm("addeo. %rt, %ra, %rb");
addeo_dot.set_decoder(op=0x1F, xo=0x8A,rc=1,oe=1);

addi.set_asm("addi %rt, %ra, %imm");
addi.set_decoder(op=0x0E);

addic.set_asm("addic %rt, %ra, %imm");
addic.set_decoder(op=0x0C);

addic_dot.set_asm("addic. %rt, %ra, %imm");
addic_dot.set_decoder(op=0x0D);

addis.set_asm("addis %rt, %ra, %imm");
addis.set_decoder(op=0x0F);

// addmex
addme.set_asm("addme %rt, %ra");
addme.set_decoder(op=0x1F, xo=0xEA, rc=0, oe=0, rb=0);

addme_dot.set_asm("addme. %rt, %ra");
addme_dot.set_decoder(op=0x1F, xo=0xEA, rc=1, oe=0, rb=0);

addmeo.set_asm("addmeo %rt, %ra");
addmeo.set_decoder(op=0x1F, xo=0xEA, rc=0, oe=1, rb=0);

addmeo_dot.set_asm("addmeo. %rt, %ra");
addmeo_dot.set_decoder(op=0x1F, xo=0xEA, rc=1, oe=1, rb=0);

// addzex
addze.set_asm("addze %rt, %ra");
addze.set_decoder(op=0x1F, xo=0xCA, rc=0, oe=0, rb=0);

addze_dot.set_asm("addze. %rt, %ra");
addze_dot.set_decoder(op=0x1F, xo=0xCA, rc=1, oe=0, rb=0);

addzeo.set_asm("addzeo %rt, %ra");
addzeo.set_decoder(op=0x1F, xo=0xCA, rc=0, oe=1, rb=0);

addzeo_dot.set_asm("addzeo. %rt, %ra");
addzeo_dot.set_decoder(op=0x1F, xo=0xCA, rc=1, oe=1, rb=0);

instr_and.set_asm("and %ra, %rt, %rb");
instr_and.set_decoder(op=0x1F, eo=0x1C, rc=0);

and_dot.set_asm("and. %ra, %rt, %rb");
and_dot.set_decoder(op=0x1F, eo=0x1C, rc=1);

andc.set_asm("andc %ra, %rt, %rb");
andc.set_decoder(op=0x1F, eo=0x3C, rc=0);
```

```
andc_dot.set_asm("andc. %ra, %rt, %rb");
andc_dot.set_decoder(op=0x1F, eo=0x3C, rc=1);

andi_dot.set_asm("andi. %ra, %rt, %imm");
andi_dot.set_decoder(op=0x1C);

andis_dot.set_asm("andis. %ra, %rt, %imm");
andis_dot.set_decoder(op=0x1D);

// Incodicional Branches
b.set_asm("b %li");
b.set_decoder(op=0x12, aa=0, rc=0);

ba.set_asm("ba %li");
ba.set_decoder(op=0x12, aa=1, rc=0);

bl.set_asm("bl %li");
bl.set_decoder(op=0x12, aa=0, rc=1);

bla.set_asm("bla %li");
bla.set_decoder(op=0x12, aa=1, rc=1);

// Codicional Branches
bc.set_asm("bc %rt, %ra, %bd");
bc.set_decoder(op=0x10, aa=0, rc=0);

bca.set_asm("bca %rt, %ra, %bd");
bca.set_decoder(op=0x10, aa=1, rc=0);

bcl.set_asm("bcl %rt, %ra, %bd");
bcl.set_decoder(op=0x10, aa=0, rc=1);

bcla.set_asm("bc %rt, %ra, %bd");
bcla.set_decoder(op=0x10, aa=1, rc=1);

bcctr.set_asm("bcctr %rt, %ra");
bcctr.set_decoder(op=0x13, eo=0x210,rb=0,rc=0);

bcctrl.set_asm("bcctrl %rt, %ra");
bcctrl.set_decoder(op=0x13, eo=0x210,rb=0,rc=1);

bclr.set_asm("bclr %rt, %ra");
bclr.set_decoder(op=0x13, eo=0x10,rb=0,rc=0);

bclrl.set_asm("bclrl %rt, %ra");
bclrl.set_decoder(op=0x13, eo=0x10,rb=0,rc=1);

cmp.set_asm("cmp %rt, 0, %ra, %rb");
```

```
cmp.set_decoder(op=0x1F, eo=0, rc=0);

cmpi.set_asm("cmpi %rt, 0, %ra, %imm");
cmpi.set_decoder(op=0x0B);

cmpl.set_asm("cmpl %rt, 0, %ra, %rb");
cmpl.set_decoder(op=0x1F, eo=0x20, rc=0);

cmpli.set_asm("cmpli %rt, 0, %ra, %imm");
cmpli.set_decoder(op=0x0A);

cntlzw.set_asm("cntlzw %ra, %rt");
cntlzw.set_decoder(op=0x1F, eo=0x1A,rb=0,rc=0);

cntlzw_dot.set_asm("cntlzw. %ra, %rt");
cntlzw_dot.set_decoder(op=0x1F, eo=0x1A,rb=0,rc=1);

crand.set_asm("crand %rt, %ra, %rb");
crand.set_decoder(op=0x13, eo=0x101,rc=0);

crandc.set_asm("crandc %rt, %ra, %rb");
crandc.set_decoder(op=0x13, eo=0x81,rc=0);

creqv.set_asm("creqv %rt, %ra, %rb");
creqv.set_decoder(op=0x13, eo=0x121,rc=0);

crnand.set_asm("crnand %rt, %ra, %rb");
crnand.set_decoder(op=0x13, eo=0xE1,rc=0);

crnor.set_asm("crnor %rt, %ra, %rb");
crnor.set_decoder(op=0x13, eo=0x21,rc=0);

cror.set_asm("cror %rt, %ra, %rb");
cror.set_decoder(op=0x13, eo=0x1C1,rc=0);

crorc.set_asm("crorc %rt, %ra, %rb");
crorc.set_decoder(op=0x13, eo=0x1A1,rc=0);

crxor.set_asm("crxor %rt, %ra, %rb");
crxor.set_decoder(op=0x13, eo=0xC1,rc=0);

// divwx
divw.set_asm("divw %rt, %ra, %rb");
divw.set_decoder(op=0x1F,xo=0x1EB,oe=0,rc=0);

divw_dot.set_asm("divw. %rt, %ra, %rb");
divw_dot.set_decoder(op=0x1F,xo=0x1EB,oe=0,rc=1);

divwo.set_asm("divwo %rt, %ra, %rb");
```

```
divwo.set_decoder(op=0x1F,xo=0x1EB,oe=1,rc=0);

divwo_dot.set_asm("divwo. %rt, %ra, %rb");
divwo_dot.set_decoder(op=0x1F,xo=0x1EB,oe=1,rc=1);

// divwux
divwu.set_asm("divuw %rt, %ra, %rb");
divwu.set_decoder(op=0x1F,xo=0x1CB,oe=0,rc=0);

divwu_dot.set_asm("divuw. %rt, %ra, %rb");
divwu_dot.set_decoder(op=0x1F,xo=0x1CB,oe=0,rc=1);

divwuo.set_asm("divuwo %rt, %ra, %rb");
divwuo.set_decoder(op=0x1F,xo=0x1CB,oe=1,rc=0);

divwuo_dot.set_asm("divuwo. %rt, %ra, %rb");
divwuo_dot.set_decoder(op=0x1F,xo=0x1CB,oe=1,rc=1);

eqv.set_asm("eqv %ra, %rt, %rb");
eqv.set_decoder(op=0x1F, eo=0x11C,rc=0);

eqv_dot.set_asm("eqv. %ra, %rt, %rb");
eqv_dot.set_decoder(op=0x1F, eo=0x11C,rc=1);

extsb.set_asm("extsb %ra, %rt");
extsb.set_decoder(op=0x1F, eo=0x3BA,rb=0,rc=0);

extsb_dot.set_asm("extsb. %ra, %rt");
extsb_dot.set_decoder(op=0x1F, eo=0x3BA,rb=0,rc=1);

extsh.set_asm("extsh %ra, %rt");
extsh.set_decoder(op=0x1F, eo=0x39A,rb=0,rc=0);

extsh_dot.set_asm("extsh. %ra, %rt");
extsh_dot.set_decoder(op=0x1F, eo=0x39A,rb=0,rc=1);

lbz.set_asm("lbz %rt, %imm(%ra)");
lbz.set_decoder(op=0x22);

lbzu.set_asm("lbzu %rt, %imm(%ra)");
lbzu.set_decoder(op=0x23);

lbzux.set_asm("lbzux %rt, %ra, %rb");
lbzux.set_decoder(op=0x1F, eo=0x77, rc=0);

lbzx.set_asm("lbzx %rt, %ra, %rb");
lbzx.set_decoder(op=0x1F, eo=0x57, rc=0);

lha.set_asm("lha %rt, %imm(%ra)");
```

```
lha.set_decoder(op=0x2A);

lhau.set_asm("lhau %rt, %imm(%ra)");
lhau.set_decoder(op=0x2B);

lhaux.set_asm("lhaux %rt, %ra, %rb");
lhaux.set_decoder(op=0x1F, eo=0x177, rc=0);

lhax.set_asm("lhax %rt, %ra, %rb");
lhax.set_decoder(op=0x1F, eo=0x157, rc=0);

lhbrx.set_asm("lhbrx %rt, %ra, %rb");
lhbrx.set_decoder(op=0x1F,eo=0x316, rc=0);

lhz.set_asm("lhz %rt, %imm(%ra)");
lhz.set_decoder(op=0x28);

lhzu.set_asm("lhzu %rt, %imm(%ra)");
lhzu.set_decoder(op=0x29);

lhzux.set_asm("lhzux %rt, %ra, %rb");
lhzux.set_decoder(op=0x1F,eo=0x137, rc=0);

lhzx.set_asm("lhzx %rt, %ra, %rb");
lhzx.set_decoder(op=0x1F,eo=0x117, rc=0);

lmw.set_asm("lmw %rt, %imm(%ra)");
lmw.set_decoder(op=0x2E);

lswi.set_asm("lswi %rt, %ra, %rb");
lswi.set_decoder(op=0x1F,eo=0x255, rc=0);

lswx.set_asm("lswx %rt, %ra, %rb");
lswx.set_decoder(op=0x1F,eo=0x215, rc=0);

lwarx.set_asm("lwarx %rt, %ra, %rb");
lwarx.set_decoder(op=0x1F,eo=0x14, rc=0);

lwbrx.set_asm("lwarx %rt, %ra, %rb");
lwbrx.set_decoder(op=0x1F,eo=0x216, rc=0);

lwz.set_asm("lwz %rt, %imm(%ra)");
lwz.set_decoder(op=0x20);

lwzu.set_asm("lwzu %rt, %imm(%ra)");
lwzu.set_decoder(op=0x21);

lwzux.set_asm("lwzux %rt, %ra, %rb");
lwzux.set_decoder(op=0x1F,eo=0x37, rc=0);
```

```
lwzx.set_asm("lwzx %rt, %ra, %rb");
lwzx.set_decoder(op=0x1F,eo=0x17, rc=0);

mcrf.set_asm("mcrf %crfD, %crfS");
mcrf.set_decoder(op=0x13, na1=0, na2=0, rb=0, eo=0, rc=0);

mcrxr.set_asm("mcrxr %crfD");
mcrxr.set_decoder(op=0x1F, crfS=0, na1=0, na2=0, rb=0, eo=0x200,rc=0);

mfcr.set_asm("mfcr %rt");
mfcr.set_decoder(op=0x1F, ra=0, rb=0, eo=0x13, rc=0);

mfmsr.set_asm("mfmsr %rt");
mfmsr.set_decoder(op=0x1F, ra=0, rb=0, eo=0x53, rc=0);

mfspr.set_asm("mfspr %rt, %spr");
mfspr.set_decoder(op=0x1F, eo=0x153, rc=0);

mftb.set_asm("mftb %rt, %spr");
mftb.set_decoder(op=0x1F, eo=0x173, rc=0);

mtcrf.set_asm("mtcrf %spr, %rt");
mtcrf.set_decoder(op=0x1F, eo=0x90);

mtmsr.set_asm("mtmsr %rt");
mtmsr.set_decoder(op=0x1F, ra=0, rb=0, eo=0x92, rc=0);

mtspr.set_asm("mtspr %spr, %rt");
mtspr.set_decoder(op=0x1F, eo=0x1D3, rc=0);

mulhw.set_asm("mulhw %rt, %ra, %rb");
mulhw.set_decoder(op=0x1F,xo=0x4B,oe=0,rc=0);

mulhw_dot.set_asm("mulhw. %rt, %ra, %rb");
mulhw_dot.set_decoder(op=0x1F,xo=0x4B,oe=0,rc=1);

mulhwu.set_asm("mulhwu %rt, %ra, %rb");
mulhwu.set_decoder(op=0x1F,xo=0xB,oe=0,rc=0);

mulhwu_dot.set_asm("mulhwu. %rt, %ra, %rb");
mulhwu_dot.set_decoder(op=0x1F,xo=0xB,oe=0,rc=1);

mulli.set_asm("mulli %rt, %ra, %imm");
mulli.set_decoder(op=0x07);

// mullwx
mullw.set_asm("mullw %rt, %ra, %rb");
mullw.set_decoder(op=0x1F,xo=0xEB,oe=0,rc=0);
```

```
mullw_dot.set_asm("mullw. %rt, %ra, %rb");
mullw_dot.set_decoder(op=0x1F,xo=0xEB,oe=0,rc=1);

mullwo.set_asm("mullwo %rt, %ra, %rb");
mullwo.set_decoder(op=0x1F,xo=0xEB,oe=1,rc=0);

mullwo_dot.set_asm("mullw %rt, %ra, %rb");
mullwo_dot.set_decoder(op=0x1F,xo=0xEB,oe=1,rc=1);

instr_nand.set_asm("nand %ra, %rt, %rb");
instr_nand.set_decoder(op=0x1F, eo=0x1DC,rc=0);

nand_dot.set_asm("nand. %ra, %rt, %rb");
nand_dot.set_decoder(op=0x1F, eo=0x1DC,rc=1);

// negx
instr_neg.set_asm("neg %rt, %ra");
instr_neg.set_decoder(op=0x1F, rb=0,xo=0x68,oe=0,rc=0);

nego.set_asm("nego %rt, %ra");
nego.set_decoder(op=0x1F, rb=0,xo=0x68,oe=1,rc=0);

neg_dot.set_asm("neg. %rt, %ra");
neg_dot.set_decoder(op=0x1F, rb=0,xo=0x68,oe=0,rc=1);

nego_dot.set_asm("nego. %rt, %ra");
nego_dot.set_decoder(op=0x1F, rb=0,xo=0x68,oe=1,rc=1);

instr_nor.set_asm("nor %ra, %rt, %rb");
instr_nor.set_decoder(op=0x1F, eo=0x7c,rc=0);

nor_dot.set_asm("nor. %ra, %rt, %rb");
nor_dot.set_decoder(op=0x1F, eo=0x7c,rc=1);

instr_or.set_asm("or %ra, %rt, %rb");
instr_or.set_decoder(op=0x1F, eo=0x1BC, rc=0);

or_dot.set_asm("or. %ra, %rt, %rb");
or_dot.set_decoder(op=0x1F, eo=0x1BC, rc=1);

orc.set_asm("orc %ra, %rt, %rb");
orc.set_decoder(op=0x1F, eo=0x19c, rc=0);

orc_dot.set_asm("orc. %ra, %rt, %rb");
orc_dot.set_decoder(op=0x1F, eo=0x19c, rc=1);

ori.set_asm("ori %ra, %rt, %imm");
ori.set_decoder(op=0x18);
```

```
oris.set_asm("oris %ra, %rt, %imm");
oris.set_decoder(op=0x19);

rfi.set_asm("rfi");
rfi.set_decoder(op=0x13,rt=0,ra=0,rb=0,eo=0x32,rc=0);

sc.set_asm("sc");
sc.set_decoder(op=0x11,rt=0,ra=0,rb=0,eo=0x01,rc=0);

rlwimi.set_asm("rlwimi %ra, %rt, %rb, %mb, %me");
rlwimi.set_decoder(op=0x14, rc=0);

rlwimi_dot.set_asm("rlwimi. %ra, %rt, %rb, %mb, %me");
rlwimi_dot.set_decoder(op=0x14, rc=1);

rlwinm.set_asm("rlwinm %ra, %rt, %rb, %mb, %me");
rlwinm.set_decoder(op=0x15, rc=0);

rlwinm_dot.set_asm("rlwinm. %ra, %rt, %rb, %mb, %me");
rlwinm_dot.set_decoder(op=0x15, rc=1);

rlwnm.set_asm("rlwnm %ra, %rt, %rb, %mb, %me");
rlwnm.set_decoder(op=0x17, rc=0);

rlwnm_dot.set_asm("rlwnm. %ra, %rt, %rb, %mb, %me");
rlwnm_dot.set_decoder(op=0x17, rc=1);

slw.set_asm("slw %ra, %rt, %rb");
slw.set_decoder(op=0x1F, eo=0x18,rc=0);

slw_dot.set_asm("slw. %ra, %rt, %rb");
slw_dot.set_decoder(op=0x1F, eo=0x18,rc=1);

sraw.set_asm("sraw %ra, %rt, %rb");
sraw.set_decoder(op=0x1F, eo=0x318,rc=0);

sraw_dot.set_asm("sraw. %ra, %rt, %rb");
sraw_dot.set_decoder(op=0x1F, eo=0x318,rc=1);

srawi.set_asm("srawi %ra, %rt, %rb");
srawi.set_decoder(op=0x1F, eo=0x338,rc=0);

srawi_dot.set_asm("srawi. %ra, %rt, %rb");
srawi_dot.set_decoder(op=0x1F, eo=0x338,rc=1);

srw.set_asm("srw %ra, %rt, %rb");
srw.set_decoder(op=0x1F, eo=0x218,rc=0);
```

```
srw_dot.set_asm("srw. %ra, %rt, %rb");
srw_dot.set_decoder(op=0x1F, eo=0x218,rc=1);

stb.set_asm("stb %rt, %imm(%ra)");
stb.set_decoder(op=0x26);

stbu.set_asm("stbu %rt, %imm(%ra)");
stbu.set_decoder(op=0x27);

stbux.set_asm("stbux %rt, %ra, %rb");
stbux.set_decoder(op=0x1F, eo=0xF7, rc=0);

stbx.set_asm("stbx %rt, %ra, %rb");
stbx.set_decoder(op=0x1F, eo=0xD7, rc=0);

sth.set_asm("sth %rt, %imm(%ra)");
sth.set_decoder(op=0x2C);

sthbrx.set_asm("sthbrx %rt, %ra, %rb");
sthbrx.set_decoder(op=0x1F,eo=0x396, rc=0);

sthu.set_asm("sthu %rt, %imm(%ra)");
sthu.set_decoder(op=0x2D);

sthux.set_asm("sthux %rt, %ra, %rb");
sthux.set_decoder(op=0x1F,eo=0x1B7, rc=0);

sthx.set_asm("sthx %rt, %ra, %rb");
sthx.set_decoder(op=0x1F,eo=0x197, rc=0);

stmw.set_asm("stmw %rt, %imm(%ra)");
stmw.set_decoder(op=0x2F);

stswi.set_asm("stswi %rt, %ra, %rb");
stswi.set_decoder(op=0x1F,eo=0x2D5, rc=0);

stswx.set_asm("stswx %rt, %ra, %rb");
stswx.set_decoder(op=0x1F,eo=0x295, rc=0);

stw.set_asm("stw %rt, %d(%ra)");
stw.set_decoder(op=0x24);

stwbrx.set_asm("stwbrx %rt, %ra, %rb");
stwbrx.set_decoder(op=0x1F,eo=0x296, rc=0);

stwcx_dot.set_asm("stwcx. %rt, %ra, %rb");
stwcx_dot.set_decoder(op=0x1F, eo=0x96, rc=0);

stwu.set_asm("stwu %rt, %imm(%ra)");
```

```
stwu.set_decoder(op=0x25);

stwux.set_asm("stwux %rt, %ra, %rb");
stwux.set_decoder(op=0x1F,eo=0xB7, rc=0);

stwx.set_asm("stwx %rt, %ra, %rb");
stwx.set_decoder(op=0x1F,eo=0x97, rc=0);

// Subx
subf.set_asm("subf %rt, %ra, %rb");
subf.set_decoder(op=0x1F, xo=0x28, rc=0, oe=0);

subf_dot.set_asm("subf. %rt, %ra, %rb");
subf_dot.set_decoder(op=0x1F, xo=0x28, rc=1, oe=0);

subfo.set_asm("subfo %rt, %ra, %rb");
subfo.set_decoder(op=0x1F, xo=0x28, rc=0, oe=1);

subfo_dot.set_asm("subfo. %rt, %ra, %rb");
subfo_dot.set_decoder(op=0x1F, xo=0x28, rc=1, oe=1);

// subfcx
subfc.set_asm("subfc %rt, %ra, %rb");
subfc.set_decoder(op=0x1F, xo=0x08, rc=0, oe=0);

subfc_dot.set_asm("subfc. %rt, %ra, %rb");
subfc_dot.set_decoder(op=0x1F, xo=0x08, rc=1, oe=0);

subfco.set_asm("subfco %rt, %ra, %rb");
subfco.set_decoder(op=0x1F, xo=0x08, rc=0, oe=1);

subfco_dot.set_asm("subfco. %rt, %ra, %rb");
subfco_dot.set_decoder(op=0x1F, xo=0x08, rc=1, oe=1);

// subfex
subfe.set_asm("subfe %rt, %ra, %rb");
subfe.set_decoder(op=0x1F, xo=0x88, rc=0, oe=0);

subfe_dot.set_asm("subfe. %rt, %ra, %rb");
subfe_dot.set_decoder(op=0x1F, xo=0x88, rc=1, oe=0);

subfeo.set_asm("subfeo %rt, %ra, %rb");
subfeo.set_decoder(op=0x1F, xo=0x88, rc=0, oe=1);

subfeo_dot.set_asm("subfeo. %rt, %ra, %rb");
subfeo_dot.set_decoder(op=0x1F, xo=0x88, rc=1, oe=1);

subfic.set_asm("subfic %rt, %ra, %imm");
subfic.set_decoder(op=0x08);
```

```
    // subfmex
    subfme.set_asm("subfme %rt, %ra, %rb");
    subfme.set_decoder(op=0x1F, xo=0xE8, rc=0, oe=0);

    subfme_dot.set_asm("subfme. %rt, %ra, %rb");
    subfme_dot.set_decoder(op=0x1F, xo=0xE8, rc=1, oe=0);

    subfmeo.set_asm("subfmeo %rt, %ra, %rb");
    subfmeo.set_decoder(op=0x1F, xo=0xE8, rc=0, oe=1);

    subfmeo_dot.set_asm("subfmeo. %rt, %ra, %rb");
    subfmeo_dot.set_decoder(op=0x1F, xo=0xE8, rc=1, oe=1);

    // subfzex
    subfze.set_asm("subfze %rt, %ra");
    subfze.set_decoder(op=0x1F, xo=0xC8, rc=0, oe=0, rb=0);

    subfze_dot.set_asm("subfze. %rt, %ra");
    subfze_dot.set_decoder(op=0x1F, xo=0xC8, rc=1, oe=0, rb=0);

    subfzeo.set_asm("subfzeo %rt, %ra");
    subfzeo.set_decoder(op=0x1F, xo=0xC8, rc=0, oe=1, rb=0);

    subfzeo_dot.set_asm("subfzeo. %rt, %ra");
    subfzeo_dot.set_decoder(op=0x1F, xo=0xC8, rc=1, oe=1, rb=0);

    instr_xor.set_asm("xor %ra, %rt, %rb");
    instr_xor.set_decoder(op=0x1F, eo=0x13c, rc=0);

    xor_dot.set_asm("xor. %ra, %rt, %rb");
    xor_dot.set_decoder(op=0x1F, eo=0x13c, rc=1);

    xori.set_asm("xori %ra, %rt, %imm");
    xori.set_decoder(op=0x1A);

    xoris.set_asm("xoris %ra, %rt, %imm");
    xoris.set_decoder(op=0x1B);

  };
};
```

5.1.3 Arquivo powerpc_syscall.cpp

```
#include "powerpc_syscall.H"
#include "ac_resources.H"

void powerpc_syscall::get_buffer(int argn, unsigned char* buf, unsigned int size)
{
  unsigned int addr = GPR.read(3+argn);
```

```
  for (unsigned int i = 0; i<size; i++, addr++) {
    buf[i] = MEM.read_byte(addr);
  }
}

void powerpc_syscall::set_buffer(int argn, unsigned char* buf, unsigned int size)
{
  unsigned int addr = GPR.read(3+argn);
  for (unsigned int i = 0; i<size; i++, addr++) {
    MEM.write_byte(addr, buf[i]);
  }
}

void powerpc_syscall::set_buffer_noinvert(int argn, unsigned char* buf, unsigned int size)
{
  unsigned int addr = GPR.read(3+argn);
  for (unsigned int i = 0; i<size; i+=4, addr+=4) {
    MEM.write(addr, *(unsigned int *) &buf[i]);
  }
}

int powerpc_syscall::get_int(int argn)
{
  return GPR.read(3+argn);
}

void powerpc_syscall::set_int(int argn, int val)
{
  GPR.write(3+argn, val);
}

void powerpc_syscall::return_from_syscall()
{
  GPR.write(1, MEM.read(GPR.read(31)));
  GPR.write(31, MEM.read(GPR.read(31) + 28));
  ac_pc = LK;
}

void powerpc_syscall::set_prog_args(int argc, char **argv)
{
  extern unsigned AC_RAM_END;
  int i, j, base;
  unsigned int ac_argv[30];
  char ac_argstr[512];

  base = AC_RAM_END - 512;
  for (i=0, j=0; i<argc; i++) {
    int len = strlen(argv[i]) + 1;
    ac_argv[i] = base + j;
```

```
    memcpy(&ac_argstr[j], argv[i], len);
    j += len;
  }

  //Ajust %sp and write argument string
  GPR.write(1, AC_RAM_END-512);
  set_buffer(-2, (unsigned char*) ac_argstr, 512);   //$-3 = $1(sp) - 4(set_buffer adds 4)

  //Ajust %sp and write string pointers
  GPR.write(1, AC_RAM_END-512-120);
  set_buffer_noinvert(-2, (unsigned char*) ac_argv, 120);

  //Set %o0 to the argument count
  GPR.write(3, argc);

  //Set %o1 to the string pointers
  GPR.write(4, AC_RAM_END-512-120);

}
```

5.1.4 Arquivo powerpc-isa.cpp

```
/*****************************************************
 *   Arquivo de descrição de comportamento do PowerPC  *
 *   Autores:                                          *
 *   Daniel Carlos Casarotto                           *
 *   José Otávio Carlomagno Filho                      *
 *****************************************************/

#include  "powerpc-isa.H"

//ISA initialization
#include "ac_isa_init.cpp"

//------------------------------------------------------------------------------
//#define DEBUG // Comente essa linha para não exibir informações de debug
//#define REGISTER_DEBUG
//#define ACPC
//#define IMPRIME_INSTRUCOES
#ifdef DEBUG
  // Utilize o método imprime para imprimir na tela as informações de debug
  #define imprime(str, args...) printf(str , args)
  #define imprime_string(str) printf(str)
#else
  #define imprime(str, args...)
  #define imprime_string(str);
#endif

#ifdef IMPRIME_INSTRUCOES
  #define printInstruction(str, args...) printf(str , args)
```

```
#else
  #define printInstruction(str, args...)
#endif

#define UINT64 unsigned long long
#define SINT64 long long

void printRegisterStatus()
{
#ifdef REGISTER_DEBUG
  printf("=========== Register Status ============\n");
  printf("R00 :0x%X\tR01 :0x%X\tR02 :0x%X\tR03 :0x%X\tR04 :0x%X\n",GPR.read(0),
    GPR.read(1),GPR.read(2),GPR.read(3),GPR.read(4));
  printf("R05 :0x%X\tR06 :0x%X\tR07 :0x%X\tR08 :0x%X\tR09 :0x%X\n",GPR.read(5),
    GPR.read(6),GPR.read(7),GPR.read(8),GPR.read(9));
  printf("R10 :0x%X\tR11 :0x%X\tR12 :0x%X\tR13 :0x%X\tR14 :0x%X\n",GPR.read(10),
    GPR.read(11),GPR.read(12),GPR.read(13),GPR.read(14));
  printf("R15 :0x%X\tR16 :0x%X\tR17 :0x%X\tR18 :0x%X\tR19 :0x%X\n",GPR.read(15),
    GPR.read(16),GPR.read(17),GPR.read(18),GPR.read(19));
  printf("R20 :0x%X\tR21 :0x%X\tR22 :0x%X\tR23 :0x%X\tR24 :0x%X\n",GPR.read(20),
    GPR.read(21),GPR.read(22),GPR.read(23),GPR.read(24));
  printf("R25 :0x%X\tR26 :0x%X\tR27 :0x%X\tR28 :0x%X\tR29 :0x%X\n",GPR.read(25),
    GPR.read(26),GPR.read(27),GPR.read(28),GPR.read(29));
  printf("R30 :0x%X\tR31 :0x%X\tCTR :0x%X\tLK  :0x%X\tCR  :0x%X\n",GPR.read(30),
    GPR.read(31),(int)CTR,(int)LK,(int)CR);
  printf("XER[ca] :0x%x\n",(int)XER.ca);
  printf("=========== Register Status ============\n");
#endif
}
//-------------------------------------------------------------------------
//!Generic instruction behavior method.
void ac_behavior( instruction ){
  imprime("\n\nPC= 0x%x\tInstruction counter: %d\n", (int) ac_pc, (int)ac_instr_counter);
#ifdef ACPC
  printf("%x ou %d\n", (int)ac_pc, (int)ac_pc);
#endif
  ac_pc = ac_pc +4;
  printRegisterStatus();
};
 //-------------------------------------------------------------------------

//! Instruction Format behavior methods.
void ac_behavior( Type_XO ){}
void ac_behavior( Type_D ){}
void ac_behavior( Type_X ){}
void ac_behavior( Type_I ){}
void ac_behavior( Type_B ){}
void ac_behavior( Type_M ){}
void ac_behavior( Type_XL ){}
```

```
void ac_behavior( Type_XFX ){}

//------------------------------------------------------------------------------

inline void SetXERAddOverflow(int a, int b, int resultado)
{
  bool overflow = false;
  if (a>=0 && b>= 0 && resultado <0)
    overflow = true;
  else if (a<0 && b<0 && resultado >=0)
    overflow = true;

  if (overflow){
    XER.ov = 1;  // Bit de overflow, setado sempre quando ocorre overflow
    // Bit de sumário, setado sempre que bit OV é setado, e permanece até ser
    // limpo por uma instrução mtspr ou mcrxr
    XER.so = 1;
  } else
    XER.ov = 0;
}
//------------------------------------------------------------------------------
inline void SetXERSubOverflow(int a, int b, int resultado)
{
  bool overflow = false;
  if (a>=0 && b>= 0 && resultado <0)
    overflow = true;
  else if (a<0 && b<0 && resultado >=0)
    overflow = true;

  if (overflow){
    XER.ov = 1;  // Bit de overflow, setado sempre quando ocorre overflow
    // Bit de sumário, setado sempre que bit OV é setado, e permanece até ser
    // limpo por uma instrução mtspr ou mcrxr
    XER.so = 1;
  } else
    XER.ov = 0;
}
//------------------------------------------------------------------------------
inline void atualizaCR0(int resultado)
{
  // Limpa campo 0 antes
  CR = CR & 0x0FFFFFFF;
  if (resultado < 0)
    CR = CR | 0x80000000;
  else if (resultado == 0)
    CR = CR | 0x20000000;
  else // > 0
    CR = CR | 0x40000000;
```

```
  if (XER.so)
    CR = CR | 0x10000000;
}
//-----------------------------------------------------------------------------
inline void GenericBranch(
  bool efectiveAddr,
  bool link,
  unsigned bo,
  unsigned bi,
  unsigned bd
)
{
  if (link)
    LK = ac_pc;

  unsigned int bo0 = (bo & 0x10)>>4;
  unsigned int bo1 = (bo & 0x8)>>3;
  unsigned int bo2 = (bo & 0x4)>>2;
  unsigned int bo3 = (bo & 0x2)>>1;

  if (bo2 == 0)
    CTR = CTR -1;

  unsigned crbi = (CR >>(31 - bi)) & 0x1;

  if ((bo2==1 || ((CTR==0)==bo3)) && (bo0 == 1 || (crbi == bo1)))
  {
    if(efectiveAddr)
      ac_pc = (bd<<2)-4;
    else
      ac_pc += (bd<<2)-4;

  }
}
//-----------------------------------------------------------------------------
inline int CarryOut(unsigned long long result)
{
  unsigned max = 0xFFFFFFFF;
  if(result > max){
    return 1;
  } else
    return 0;

}
//-----------------------------------------------------------------------------
inline void GenericLogicCompare(unsigned cf, unsigned ra, unsigned rb)
{
  unsigned int mascara;
  if (ra < rb)
```

```
    mascara = 0x80000000;
  else if (ra > rb)
    mascara = 0x40000000;
  else
    mascara = 0x20000000;
  if(XER.so)
    mascara = mascara | 0x10000000;

  unsigned int posicao = cf *4;
  unsigned int mascara2 = 0xF0000000;
  mascara = mascara >> posicao;
  mascara2 = mascara2 >> posicao;
  CR = CR & ~mascara2; // Zerar campo
  CR = CR | mascara; // Setar Bits

}
//------------------------------------------------------------------------------

inline void GenericCompare(unsigned cf, int ra, int rb)
{
  unsigned int mascara;
  if (ra < rb)
    mascara = 0x80000000;
  else if (ra > rb)
    mascara = 0x40000000;
  else
    mascara = 0x20000000;
  if(XER.so)
    mascara = mascara | 0x10000000;

  unsigned int posicao = cf * 4;
  unsigned int mascara2 = 0xF0000000;
  mascara = mascara >> posicao;
  mascara2 = mascara2 >> posicao;
  CR = CR & ~mascara2; // Zerar campo
  CR = CR | mascara; // Setar Bits


}

//------------------------------------------------------------------------------
inline unsigned rotl(unsigned rs, unsigned n)
{
  unsigned ret = (rs << n) | ((rs >> (32 - n)) & ((1 << n) - 1));
  return ret;
}
//------------------------------------------------------------------------------
inline unsigned mask(unsigned mb, unsigned me)
{
```

```
  unsigned mask = (((mb > me) ? ~((0xFFFFFFFF >> mb) ^
    ((me >= 31) ? 0 : 0xFFFFFFFF >> (me + 1))):
            ((0xFFFFFFFF >> mb) ^ ((me >= 31) ? 0 : 0xFFFFFFFF >> (me + 1)))));

  return mask;
}
//------------------------------------------------------------------------------
void ac_behavior( addi )
{
  printInstruction("addi %d, %d, %d\n", rt, ra, imm & 0xFFFF);
  unsigned valorRA = ra == 0 ? 0 : GPR.read(ra);
  imprime("Operacao -> rt = %d + %d\n",valorRA,imm);
  GPR.write(rt, valorRA + imm);
  imprime("Resultado ->  rt = %d\n", GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior( add )
{
  printInstruction("add %d, %d, %d\n",rt,ra,rb);
  imprime("Operacao -> rt = %d + %d\n", GPR.read(ra),GPR.read(rb));
  GPR.write(rt, GPR.read(ra) + GPR.read(rb));
  imprime("Resultado -> rt = %d\n", GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior( subf )
{
  printInstruction("subf %d, %d, %d\n",rt,ra,rb);
  imprime("Operacao -> rt = %d - %d\n", GPR.read(rb),GPR.read(ra));
  GPR.write(rt, GPR.read(rb) -  GPR.read(ra));
  imprime("Resultado -> rt = %d\n", GPR.read(rt));

}
//------------------------------------------------------------------------------
void ac_behavior( lwz )
{
  printInstruction("lwz %d, %d(%d)\n",rt,imm,ra);
  imprime("Reg. %d <- conteudo da pos. de mem. %d + %d\n",rt,imm,GPR.read(ra));
  unsigned valor = ra == 0 ? 0 : GPR.read(ra);
  GPR.write(rt, MEM.read( valor+imm));
  imprime("Resultado: %d\n",GPR.read(rt));

}
//------------------------------------------------------------------------------
void ac_behavior( stw )
{
  printInstruction("stw %d, %d(%d)\n",rt,imm,ra);
  unsigned valor = ra == 0 ? 0 : GPR.read(ra);
  imprime("MEM[%d] <-  %d\n",valor+imm,GPR.read(rt));
```

```
  MEM.write( valor+imm, GPR.read(rt));
  imprime("Resultado: %d\n",MEM.read(valor+imm));
}
//------------------------------------------------------------------------
void ac_behavior( instr_and )
{
  printInstruction("and %d, %d, %d\n",ra,rt,rb);
  imprime("Reg. %d <- %d AND %d\n",ra, GPR.read(rt), GPR.read(rb));
  GPR.write(ra, GPR.read(rb) & GPR.read(rt));
  imprime("Resultado: %d\n", GPR.read(ra));

}
//------------------------------------------------------------------------
void ac_behavior( instr_or )
{
  printInstruction("or %d, %d, %d\n",ra,rt,rb);
  imprime("Reg. %d <- %d OR %d\n",ra, GPR.read(rt), GPR.read(rb));
  GPR.write(ra, GPR.read(rb) | GPR.read(rt));
  imprime("Resultado: %d\n", GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior( or_dot )
{
  printInstruction("or. %d, %d, %d\n",ra,rt,rb);
  imprime("Reg. %d <- %d OR %d\n",ra, GPR.read(rt), GPR.read(rb));
  GPR.write(ra, GPR.read(rb) | GPR.read(rt));
  atualizaCR0(GPR.read(ra));
  imprime("Resultado: %d\n", GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior( b )
{
  printInstruction("b %d\n",li);
  imprime("Pula para endereço: %d + %d\n", (int)ac_pc, (li << 2)-4);
  ac_pc += (li<<2)-4;
  imprime("Resultado: Endereco atual %d\n",(int)ac_pc);
}
//------------------------------------------------------------------------
void ac_behavior( bc )
{
  printInstruction("bc %d, %d, %d\n",rt, ra, bd);
  GenericBranch(false, false,rt, ra, bd);
}
//------------------------------------------------------------------------
void ac_behavior( cmp )
{
  unsigned bf = (rt & 0x1C) >> 2;
  printInstruction("cmp %d, 0, %d, %d\n", bf, ra, rb);
  GenericCompare(bf, GPR.read(ra),GPR.read(rb));
```

```
}
//------------------------------------------------------------------------------
void ac_behavior(instr_nand)
{
  printInstruction("nand %d, %d, %d\n", ra, rt, rb);
  GPR.write(ra, ~(GPR.read(rt) & GPR.read(rb)));
  imprime("Resultado r%d <- %d\n",ra, GPR.read(ra));
}
//------------------------------------------------------------------------------
void ac_behavior(nand_dot)
{
  printInstruction("nand. %d, %d, %d\n", ra, rt, rb);
  GPR.write(ra, ~(GPR.read(rt) & GPR.read(rb)));
  atualizaCR0(GPR.read(ra));
  imprime("Resultado r%d <- %d\n",ra, GPR.read(ra));
}
//------------------------------------------------------------------------------
void ac_behavior(instr_neg)
{
  printInstruction("neg %d, %d\n", rt, ra);
  GPR.write(rt, ~GPR.read(ra) + 1);
  imprime("Resultado r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior(neg_dot)
{
  printInstruction("neg. %d, %d\n", rt, ra);
  GPR.write(rt, ~GPR.read(ra) + 1);
  atualizaCR0(GPR.read(rt));
  imprime("Resultado r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior(nego) // PENDENTE
{
  printInstruction("nego %d, %d\n", rt, ra);
  GPR.write(rt, ~GPR.read(ra) + 1);
  imprime("Resultado r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior(nego_dot) // PENDENTE
{
  printInstruction("nego. %d, %d\n", rt, ra);
  GPR.write(rt, ~GPR.read(ra) + 1);
  atualizaCR0(GPR.read(rt));
  imprime("Resultado r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------------

void ac_behavior(instr_nor)
```

```
{
  printInstruction("nor %d, %d %d\n", rt, ra, rb);
  GPR.write(ra, ~(GPR.read(rt) | GPR.read(rb)));
  imprime("Resultado r%d <- %d\n", ra, GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior(nor_dot)
{
  printInstruction("nor. %d, %d %d\n", rt, ra, rb);
  GPR.write(ra, ~(GPR.read(rt) | GPR.read(rb)));
  atualizaCR0(GPR.read(ra));
  imprime("Resultado r%d <- %d\n", ra, GPR.read(ra));
}

//------------------------------------------------------------------------
void ac_behavior (orc)
{
  printInstruction("orc %d, %d, %d\n", ra, rt, rb);
  GPR.write(ra, GPR.read(rt) | ~GPR.read(rb));
  imprime("Resultado r%d <- %d\n",ra, GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior (orc_dot)
{
  printInstruction("orc. %d, %d, %d\n", ra, rt, rb);
  GPR.write(ra, GPR.read(rt) | ~GPR.read(rb));
  atualizaCR0(GPR.read(ra));
  imprime("Resultado r%d <- %d\n",ra, GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior (instr_xor)
{
  printInstruction("xor %d, %d, %d\n", ra, rt, rb);
  GPR.write(ra, GPR.read(rt) ^ GPR.read(rb));
  imprime("Resultado: r%d <- %d\n",ra, GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior (xor_dot)
{
  printInstruction("xor. %d, %d, %d\n", ra, rt, rb);
  GPR.write(ra, GPR.read(rt) ^ GPR.read(rb));
  atualizaCR0(GPR.read(ra));
  imprime("Resultado: r%d <- %d\n",ra, GPR.read(ra));
}

//------------------------------------------------------------------------
void ac_behavior (xori)
{
  printInstruction("xori %d, %d, %d\n", ra, rt, imm & 0xFFFF);
```

```
  GPR.write(ra, GPR.read(rt) ^ (imm & 0xFFFF));
  imprime("Resultado: r%d <- %d\n",ra, GPR.read(ra));
}
//------------------------------------------------------------------------------
void ac_behavior (xoris)
{
  printInstruction("xoris %d, %d, %d\n", ra, rt, imm & 0xFFFF);
  GPR.write(ra, GPR.read(rt) ^ (imm<<16));
  imprime("Resultado: %d\n", GPR.read(ra));
}
//------------------------------------------------------------------------------
void ac_behavior (ori)
{
  printInstruction("ori %d, %d, %d\n", ra, rt, imm);
  GPR.write(ra, GPR.read(rt) | (imm & 0xFFFF));
  imprime("Resultado: %d <- %d\n",ra, GPR.read(ra));
}
//------------------------------------------------------------------------------
void ac_behavior (oris)
{
  printInstruction("oris %d, %d, %d\n", ra, rt, imm);
  GPR.write(ra, GPR.read(rt) | (imm<<16));
  imprime("Resultado: %d <- %d\n",ra, GPR.read(ra));
}
//------------------------------------------------------------------------------
void ac_behavior ( add_dot )
{
  printInstruction("add. %d, %d, %d\n", rt, ra, rb);
  GPR.write(rt, GPR.read(ra) + GPR.read(rb));
  atualizaCR0(GPR.read(rt));
  imprime("Resultado: %d <- %d\n",rt, GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior ( addo )
{
  printInstruction("addo %d, %d, %d\n", rt, ra, rb);
  GPR.write(rt, GPR.read(ra) + GPR.read(rb));
  SetXERAddOverflow(GPR.read(ra),GPR.read(rb),GPR.read(rt));
  imprime("Resultado: %d <- %d\n",rt, GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior(addo_dot)
{
  printInstruction("addo. %d, %d, %d\n", rt, ra, rb);
  GPR.write(rt, GPR.read(ra) + GPR.read(rb));
  SetXERAddOverflow(GPR.read(ra),GPR.read(rb),GPR.read(rt));
  atualizaCR0(GPR.read(rt));
  imprime("Resultado: %d <- %d\n",rt, GPR.read(rt));
}
```

```
//------------------------------------------------------------------------
void ac_behavior(addc)
{
  printInstruction("addc %d, %d, %d\n", rt, ra, rb);
  XER.ca = CarryOut(GPR.read(ra) + GPR.read(rb));
  GPR.write(rt, GPR.read(ra) + GPR.read(rb));
  imprime("Resultado: %d <- %d\n",rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(addc_dot)
{
  printInstruction("addc. %d, %d, %d\n", rt, ra, rb);
  XER.ca = CarryOut(GPR.read(ra) + GPR.read(rb));
  GPR.write(rt, GPR.read(ra) + GPR.read(rb));
  atualizaCR0(GPR.read(rt));
  imprime("Resultado: %d <- %d\n",rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(addco)
{
  printInstruction("addco %d, %d, %d\n", rt, ra, rb);
  XER.ca = CarryOut(GPR.read(ra) + GPR.read(rb));
  GPR.write(rt, GPR.read(ra) + GPR.read(rb));
  SetXERAddOverflow(GPR.read(ra),GPR.read(rb),GPR.read(rt));
  imprime("Resultado: %d <- %d\n",rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(addco_dot)
{
  printInstruction("addco. %d, %d, %d\n", rt, ra, rb);
  XER.ca = CarryOut(GPR.read(ra) + GPR.read(rb));
  GPR.write(rt, GPR.read(ra) + GPR.read(rb));
  SetXERAddOverflow(GPR.read(ra),GPR.read(rb),GPR.read(rt));
  atualizaCR0(GPR.read(rt));
  imprime("Resultado: %d <- %d\n",rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(adde)
{
  printInstruction("adde %d, %d, %d\n", rt, ra, rb);
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut(GPR.read(ra) + GPR.read(rb) + OldXERca);
  GPR.write(rt, GPR.read(ra) + GPR.read(rb) + OldXERca);
  imprime("Resultado: %d <- %d\n",rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(adde_dot)
{
  printInstruction("adde %d, %d, %d\n", rt, ra, rb);
```

```
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut(GPR.read(ra) + GPR.read(rb) + OldXERca);
  GPR.write(rt, GPR.read(ra) + GPR.read(rb) + OldXERca);
  atualizaCR0(GPR.read(rt));
  imprime("Resultado: %d <- %d\n",rt, GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior(addeo)
{
  printInstruction("addeo %d, %d, %d\n", rt, ra, rb);
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut(GPR.read(ra) + GPR.read(rb)+ OldXERca);
  GPR.write(rt, GPR.read(ra) + GPR.read(rb) + OldXERca);
  SetXERAddOverflow(GPR.read(ra),GPR.read(rb),GPR.read(rt));
  imprime("Resultado: %d <- %d\n",rt, GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior(addeo_dot)
{
  printInstruction("addeo. %d, %d, %d\n", rt, ra, rb);
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut(GPR.read(ra) + GPR.read(rb)+ OldXERca);
  GPR.write(rt, GPR.read(ra) + GPR.read(rb) + OldXERca);
  SetXERAddOverflow(GPR.read(ra),GPR.read(rb),GPR.read(rt));
  atualizaCR0(GPR.read(rt));
  imprime("Resultado: %d <- %d\n",rt, GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior(ba)
{
  printInstruction("ba %d\n",li);
  imprime("Pula para endereço: %d + %d\n", (int)ac_pc, (li << 2)-4);
  ac_pc = (li<<2)-4;
  imprime("Resultado: Endereco atual %d\n",(int)ac_pc);
}
//------------------------------------------------------------------------------
void ac_behavior(bl)
{
  printInstruction("bl %d\n",li);
  imprime("Pula para endereço: %d + %d\n", (int)ac_pc, (li << 2)-4);
  LK = ac_pc;
  ac_pc += (li<<2)-4;
  imprime("Resultado: Endereco atual %d\n",(int)ac_pc);
}
//------------------------------------------------------------------------------
void ac_behavior(bla)
{
  printInstruction("bla %d\n",li);
  imprime("Pula para endereço: %d + %d\n", (int)ac_pc, (li<<2)-4);
```

```
  LK = ac_pc;
  ac_pc = (li<<2)-4;
  imprime("Resultado: Endereco atual %d\n",(int)ac_pc);
}
//------------------------------------------------------------------------------
void ac_behavior( bca )
{
  printInstruction("bca %d, %d, %d\n",rt, ra, bd);
  GenericBranch(true,false,rt, ra, bd);
}
//------------------------------------------------------------------------------
void ac_behavior( bcl )
{
  printInstruction("bcl %d, %d, %d\n",rt, ra, bd);
  GenericBranch(false,true,rt, ra, bd);
}
//------------------------------------------------------------------------------
void ac_behavior( bcla )
{
  printInstruction("bcla %d, %d, %d\n",rt, ra, bd);
  GenericBranch(true,true,rt, ra, bd);
}
//------------------------------------------------------------------------------
// MACRO UTILIZADA NAS INSTRUÇÕES ABAIXO
//(crand, crandc, creqv, crnand, crnor, cror, crorc, crxor)
#define CR_BIT_OPERATION(rt, ra, rb, operatorA, operatorB, operatorC) \
  unsigned int bit1 = CR >> 31 - (ra);     \
  unsigned int bit2 = CR >> 31 - (rb);     \
  bit1 = bit1 & 1;                         \
  bit2 = bit2 & 1;                         \
  unsigned int mask = 0x80000000;          \
  mask = mask >> (rt);                     \
  bit1 = operatorA ((bit1) operatorB (operatorC (bit2))); \
  if (bit1)                                \
    CR = CR | mask;                        \
  else                                     \
    CR = CR & ~mask;                       \
//------------------------------------------------------------------------------
void ac_behavior( crand )
{
  printInstruction("crand %d, %d, %d\n",rt,ra,rb);
  CR_BIT_OPERATION(rt, ra, rb, ,&, );
}
//------------------------------------------------------------------------------
void ac_behavior( crandc )
{
  printInstruction("crandc %d, %d, %d\n",rt,ra,rb);
  CR_BIT_OPERATION(rt, ra, rb, ,&,~);
}
```

```
//------------------------------------------------------------------------
void ac_behavior( creqv )
{
  printInstruction("creqv %d, %d, %d\n",rt,ra,rb);
  CR_BIT_OPERATION(rt, ra, rb,~,^, );
}
//------------------------------------------------------------------------
void ac_behavior( crnand )
{
  printInstruction("crnand %d, %d, %d\n",rt,ra,rb);
  CR_BIT_OPERATION(rt, ra, rb,~,&, );
}
//------------------------------------------------------------------------
void ac_behavior( crnor )
{
  printInstruction("crnor %d, %d, %d\n",rt,ra,rb);
  CR_BIT_OPERATION(rt, ra, rb,~,|, );
}
//------------------------------------------------------------------------
void ac_behavior( cror )
{
  printInstruction("cror %d, %d, %d\n",rt,ra,rb);
  CR_BIT_OPERATION(rt, ra, rb, ,|, );
}
//------------------------------------------------------------------------
void ac_behavior( crorc )
{
  printInstruction("crorc %d, %d, %d\n",rt,ra,rb);
  CR_BIT_OPERATION(rt, ra, rb, ,|,~);
}
//------------------------------------------------------------------------
void ac_behavior( crxor )
{
  printInstruction("crxor %d, %d, %d\n",rt,ra,rb);
  CR_BIT_OPERATION(rt, ra, rb, ,^, );
}
//------------------------------------------------------------------------
void ac_behavior(addis)
{
  printInstruction("addis %d, %d, %d\n",rt, ra, imm);
  if (ra==0)
    GPR.write(rt, imm << 16);
  else
    GPR.write(rt, GPR.read(ra) + (imm << 16));
  imprime("Resultado: r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(and_dot)
{
```

```
  printInstruction("and. %d, %d, %d\n",ra, rt, rb);
  GPR.write(ra, GPR.read(rt) & GPR.read(rb));
  atualizaCR0(GPR.read(ra));
  imprime("Resultado: r%d <- %d\n", ra, GPR.read(ra));
}
//-----------------------------------------------------------------------------
void ac_behavior(andc)
{
  printInstruction("andc %d, %d, %d\n",ra, rt, rb);
  GPR.write(ra, GPR.read(rt) & (~GPR.read(rb)));
  imprime("Resultado: r%d <- %d\n", ra, GPR.read(ra));
}
//-----------------------------------------------------------------------------
void ac_behavior(andc_dot)
{
  printInstruction("andc. %d, %d, %d\n",ra, rt, rb);
  GPR.write(ra, GPR.read(rt) & (~GPR.read(rb)));
  atualizaCR0(GPR.read(ra));
  imprime("Resultado: r%d <- %d\n", ra, GPR.read(ra));
}
//-----------------------------------------------------------------------------
void ac_behavior(andi_dot)
{
  printInstruction("andi. %d, %d, %d\n",ra, rt, imm);
  GPR.write(ra, GPR.read(rt) & (0x0000FFFF & imm));
  atualizaCR0(GPR.read(ra));
  imprime("Resultado: r%d <- %d\n", ra, GPR.read(ra));
}
//-----------------------------------------------------------------------------
void ac_behavior(andis_dot)
{
  printInstruction("andis. %d, %d, %d\n",ra, rt, imm);
  GPR.write(ra, GPR.read(rt) & (imm << 16));
  atualizaCR0(GPR.read(ra));
  imprime("Resultado: r%d <- %d\n", rt, GPR.read(rt));
}
//-----------------------------------------------------------------------------
void ac_behavior(eqv)
{
  printInstruction("eqv %d, %d, %d\n",ra, rt, rb);
  GPR.write(ra, ~(GPR.read(rt)^GPR.read(rb)));
  imprime("Resultado: r%d <- %d\n", ra, GPR.read(ra));
}
//-----------------------------------------------------------------------------
void ac_behavior(eqv_dot)
{
  printInstruction("eqv. %d, %d, %d\n",ra, rt, rb);
  GPR.write(ra, ~(GPR.read(rt)^GPR.read(rb)));
  atualizaCR0(GPR.read(ra));
```

```
  imprime("Resultado: r%d <- %d\n", ra, GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior(stb)
{
  printInstruction("stb %d, %d(%d)\n", rt, imm, ra);
  unsigned end;
  if (ra==0)
    end = imm;
  else
    end = GPR.read(ra) + imm;
  unsigned char byte = (GPR.read(rt)&0xFF);
  MEM.write_byte(end,byte);
  imprime("Resultado: MEM[%d] <- %d\n", end, MEM.read_byte(end));
}
//------------------------------------------------------------------------
void ac_behavior(stbu)
{
  printInstruction("stbu %d, %d(%d)\n", rt, imm, ra);
  if (ra==0) {
    imprime_string("instrucao invalida, ra=0");
    return;
  }
  unsigned end = GPR.read(ra) + imm;
  unsigned char byte = (GPR.read(rt)&0xFF);
  MEM.write_byte(end,byte);
  GPR.write(ra, end);

  imprime("Resultado: MEM[%d] <- %d\n", end, MEM.read_byte(end));
  imprime("\t: r%d <- %d\n", ra, end);
}
//------------------------------------------------------------------------
void ac_behavior(stbux)
{
  printInstruction("stbux %d, %d, %d\n", rt, ra, rb);
  if (ra==0){
    imprime_string("instrucao invalida, ra=0");
    return;
  }
  unsigned end = GPR.read(ra) + GPR.read(rb);
  unsigned char byte = (GPR.read(rt)&0xFF);
  MEM.write_byte(end,byte);
  GPR.write(ra, end);
  imprime("Resultado: MEM[%d] <- %d\n", end, MEM.read_byte(end));
  imprime("\t: r%d <- %d\n", ra, end);
}
//------------------------------------------------------------------------
void ac_behavior(stbx)
{
```

```
  printInstruction("stbx %d, %d, %d\n", rt, ra, rb);
  unsigned end;
  if (ra==0)
    end = GPR.read(rb);
  else
    end = GPR.read(ra) + GPR.read(rb);

  unsigned char byte = (GPR.read(rt)&0xFF);
  MEM.write_byte(end,byte);

  imprime("Resultado: MEM[%d] <- %d\n", end, MEM.read_byte(end));
  imprime("\t: r%d <- %d\n", ra, end);
}
//------------------------------------------------------------------------------
void ac_behavior(sth)
{
  printInstruction("sth %d, %d(%d)\n", rt, imm, ra);
  unsigned end;
  if (ra==0)
    end = imm;
  else
    end = GPR.read(ra) + imm;
  unsigned short halfWord = (GPR.read(rt)&0xFFFF);
  MEM.write_half(end,halfWord);
  imprime("Resultado: MEM[%d] <- %d\n", end,MEM.read_half(end));
}
//------------------------------------------------------------------------------
void ac_behavior(lbz)
{
  printInstruction("lbz %d, %d(%d)\n", rt, imm, ra);
  unsigned end;
  if (ra==0)
    end = imm;
  else
    end = GPR.read(ra) + imm;
  GPR.write(rt, 0x000000FF & MEM.read_byte(end));

  imprime("Resultado: r%d <- %d\n", rt,GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior(lbzu)
{
  printInstruction("lbzu %d, %d(%d)\n", rt, imm, ra);
  unsigned ea = ra == 0 ? imm : GPR.read(ra) + imm;
  GPR.write(rt, 0x000000FF & MEM.read_byte(ea));
  GPR.write(ra, ea);
  imprime("Resultado: r%d <- %d -- r%d <-- %d\n", rt,GPR.read(rt), ra, ea);
}
//------------------------------------------------------------------------------
```

```
void ac_behavior(lbzux)
{
  printInstruction("lbzux %d, %d, %d\n", rt, ra, rb);
  if (ra==rt){
    imprime_string("instrucao invalida");
  return;
  }
  unsigned ea = ra == 0 ? GPR.read(rb) :  GPR.read(ra) + GPR.read(rb);
  GPR.write(rt, 0x000000FF & MEM.read_byte(ea));
  GPR.write(ra, ea);
  imprime("Resultado: r%d <- %d -- r%d <-- %d\n", rt,GPR.read(rt), ra, ea);
}
//------------------------------------------------------------------------
void ac_behavior(lbzx)
{
  printInstruction("lbzx %d, %d, %d\n", rt, ra, rb);
  unsigned end;
  if (ra==0)
    end = GPR.read(rb);
  else
    end = GPR.read(ra) + GPR.read(rb);
  GPR.write(rt, 0x000000FF & MEM.read_byte(end));

  imprime("Resultado: r%d <- %d\n", rt,GPR.read(rt));

}
//------------------------------------------------------------------------
void ac_behavior(lha)
{
  printInstruction("lha %d, %d(%d)\n", rt, imm, ra);
  unsigned end;
  if (ra==0)
    end = imm;
  else
    end = GPR.read(ra) + imm;

  short half = MEM.read_half(end); // Extender
  int word = half;
  GPR.write(rt, word);
  imprime("Resultado: r%d <- %d\n", rt,GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(lhau)
{
  printInstruction("lhau %d, %d(%d)\n", rt, imm, ra);
  if (ra==0 || ra==rt){
    imprime_string("instrucao invalida");
    return;
  }
```

```
  unsigned ea = GPR.read(ra) + imm;
  short half = MEM.read_half(ea); // Extender
  int word = half;
  GPR.write_byte(rt, word);
  GPR.write(ra, ea);
  imprime("Resultado: r%d <- %d -- r%d <-- %d\n", rt,GPR.read(rt), ra, ea);
  printf("TESTE lhau\n");

}
//------------------------------------------------------------------------------
void ac_behavior(lhaux)
{
  printInstruction("lhaux %d, %d, %d\n", rt, ra, rb);
  if (ra==0 || ra==rt){
    imprime_string("instrucao invalida");
    return;
  }
  unsigned ea = GPR.read(ra) + GPR.read(rb);
  short half = MEM.read_half(ea); // Extender
  int word = half;
  GPR.write(rt, word);
  GPR.write(ra, ea);
  imprime("Resultado: r%d <- %d -- r%d <-- %d\n", rt,GPR.read(rt), ra, ea);
  printf("TESTE lhaux\n");

}
//------------------------------------------------------------------------------
void ac_behavior(lhax)
{
  printInstruction("lhax %d, %d, %d\n", rt, ra, rb);
  unsigned end;
  if (ra==0)
    end = GPR.read(rb);
  else
    end = GPR.read(ra) + GPR.read(rb);

  short half = MEM.read_half(end); // Extender
  int word = half;
  GPR.write(rt, word);
  imprime("Resultado: r%d <- %d\n", rt,GPR.read(rt));
  printf("TESTE lhax\n");
}
//------------------------------------------------------------------------------
void ac_behavior(sthbrx)
{
  printInstruction("sthbrx %d, %d, %d\n", rt, ra, rb);
  unsigned end;
  if (ra==0)
    end = GPR.read(rb);
```

```
  else
    end = GPR.read(ra) + GPR.read(rb);
  unsigned int registrador = GPR.read(rt);
  unsigned short resultado = (registrador << 8) & 0xFF00;
  unsigned short temp = (registrador >> 8) & 0xFF;
  resultado = resultado | temp;
  MEM.write_half(end, resultado);
  imprime("Resultado: MEM[%d] <- %d\n", end,MEM.read_half(end));
  printf("TESTE sthbrx\n");
}
//------------------------------------------------------------------------
void ac_behavior(sthu)
{
  printInstruction("sthu %d, %d(%d)\n", rt, imm, ra);
  if (ra==0){
    imprime_string("instrucao invalida");
    return;
  }
  unsigned end = GPR.read(ra) + imm;
  MEM.write_half(end,GPR.read(rt)&0xFFFF);
  GPR.write(ra, end);
  imprime("Resultado: MEM[%d] <- %d -- r%d <-- %d\n", end,MEM.read_half(end),ra, GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior(sthux)
{
  printInstruction("sthux %d, %d, %d\n", rt, ra, rb);
  if (ra==0){
    imprime_string("instrucao invalida");
    return;
  }
  unsigned end = GPR.read(ra) + GPR.read(rb);
  MEM.write_half(end,GPR.read(rt)&0xFFFF);
  GPR.write(ra, end);
  imprime("Resultado: MEM[%d] <- %d -- r%d <-- %d\n", end,MEM.read_half(end),ra, GPR.read(ra));

}
//------------------------------------------------------------------------
void ac_behavior(sthx)
{
  printInstruction("sthx %d, %d, %d\n", rt, ra, rb);
  unsigned end;
  if (ra==0)
    end = GPR.read(rb);
  else
    end = GPR.read(ra) + GPR.read(rb);
  MEM.write_half(end,GPR.read(rt)&0xFFFF);
  imprime("Resultado: MEM[%d] <- %d\n", end,MEM.read_half(end));
```

```
}
//------------------------------------------------------------------------

void ac_behavior(stmw)
{
  printInstruction("stmw %d, %d(%d)\n", rt, imm, ra);
  unsigned end;
  if (ra==0)
    end = 0;
  else
    end = GPR.read(ra) + imm;
  unsigned int r = rt;
  while(r<32){
    MEM.write(end, GPR.read(r));
    r++;
    end += 4;
  }
  // PENDENTE - DSI Exception ???
  printf("TESTE");
}

//------------------------------------------------------------------------
void ac_behavior(stswi)
{
  printInstruction("stswi %d, %d, %d\n", rt, ra, rb);
  unsigned end =  ra==0 ? 0 : GPR.read(ra);
  unsigned n = rb==0 ? 32 : rb;

  unsigned r = rt - 1;
  unsigned i = 0;
  unsigned mask = 0xFF000000;
  while(n>0){
    if (i==0)
      r = (r + 1)%32;
      unsigned valor = GPR.read(r);
      valor = valor & mask;
      valor = valor >> 24 - i;
      MEM.write_byte(end, valor);
      i += 8;
      mask = mask >> 8;
      if (i==32) i=0;
      end++;
      n--;
  }
  // PENDENTE - DSI Exception ???
  printf("TESTE");
}
//------------------------------------------------------------------------
void ac_behavior(stswx)
```

```
{
  printInstruction("stswx %d, %d, %d\n", rt, ra, rb);

  unsigned end =  ra==0 ? GPR.read(rb) : GPR.read(rb) + GPR.read(ra);
  unsigned n = XER.bc;

  unsigned r = rt - 1;
  unsigned i = 0;
  unsigned mask = 0xFF000000;
  while(n>0){
    if (i==0)
      r = (r + 1)%32;
      unsigned valor = GPR.read(r);
      valor = valor & mask;
      valor = valor >> 24 - i;
      MEM.write_byte(end, valor);
      i += 8;
      mask = mask >> 8;
      if (i==32) i=0;
      end++;
      n--;
  }
  printf("TESTE");

}
//------------------------------------------------------------------------------
void ac_behavior(stwbrx)
{
  printInstruction("stwbrx %d, %d, %d\n", rt, ra, rb);
  unsigned end = ra == 0 ? GPR.read(rb) : GPR.read(ra) + GPR.read(rb);
  unsigned registrador = GPR.read(rt);
  unsigned resultado = (registrador & 0xFF) << 24;
  resultado = resultado | ((registrador & 0xFF00) << 8);
  resultado = resultado | ((registrador & 0xFF0000) >> 8);
  resultado = resultado | ((registrador & 0xFF000000) >> 24);
  MEM.write(end, resultado);
  imprime("Resultado: MEM[%d] <- %d\n", end,MEM.read(end));
  printf("TESTE");
}
//------------------------------------------------------------------------------
void ac_behavior(stwu)
{
  printInstruction("stwu %d, %d(%d)\n", rt, imm, ra);
  if (ra==0){
    imprime_string("instrucao invalida");
    return;
  }
  unsigned end = GPR.read(ra) + imm;
  MEM.write(end, GPR.read(rt));
```

```
  GPR.write(ra, end);

  imprime("Resultado: MEM[%d] <- %d -- r%d <-- %d\n", end,MEM.read(end),ra, GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior(stwux)
{
  printInstruction("stwux %d, %d, %d\n", rt, ra, rb);
  if (ra==0){
    imprime_string("instrucao invalida");
    return;
  }
  unsigned end = GPR.read(ra) + GPR.read(rb);
  MEM.write(end, GPR.read(rt));
  GPR.write(ra, end);

  imprime("Resultado: MEM[%d] <- %d -- r%d <-- %d\n", end,MEM.read(end),ra, GPR.read(ra));

}
//------------------------------------------------------------------------
void ac_behavior(stwx)
{
  printInstruction("stwx %d, %d, %d\n", rt, ra, rb);
  unsigned end = ra == 0 ? GPR.read(rb) : GPR.read(ra) + GPR.read(rb);
  MEM.write(end, GPR.read(rt));
  imprime("Resultado: MEM[%d] <- %d\n", end,MEM.read(end));
}
//------------------------------------------------------------------------
void ac_behavior(lhbrx)
{
  printInstruction("lhbrx %d, %d, %d\n", rt, ra, rb);
  unsigned end = ra==0 ? GPR.read(rb) : GPR.read(ra) + GPR.read(rb);
  unsigned valor = 0x0000FF00 & (MEM.read_byte(end+1)<<8);
  valor = valor ||  (MEM.read_byte(end+1)&0xFF);
  GPR.write(rt, valor);
  imprime("Resultado: r%d <-- %d\n",rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(lhz)
{
  printInstruction("lhz %d, %d(%d)\n", rt, imm, ra);
  unsigned end = ra==0 ? imm : GPR.read(ra) + imm;
  GPR.write(rt, 0xFFFF & MEM.read_half(end));
  imprime("Resultado: r%d <-- %d\n",rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(lhzu)
{
  printInstruction("lhzu %d, %d(%d)\n", rt, imm, ra);
```

```
  if (ra == 0 || ra==rt){
    imprime_string("instrucao invalida");
    return;
  }
  unsigned end = GPR.read(ra) + imm;
  GPR.write(rt, 0xFFFF & MEM.read_half(end));
  GPR.write(ra,end);
  imprime("Resultado: r%d <-- %d -- r%d <-- %d\n",rt, GPR.read(rt), ra, GPR.read(ra));
}
//------------------------------------------------------------------------------
void ac_behavior(lhzux)
{
  printInstruction("lhzux %d, %d, %d\n", rt, ra, rb);
  if (ra == 0 || ra==rt){
    imprime_string("instrucao invalida");
    return;
  }
  unsigned end = GPR.read(ra) + GPR.read(rb);
  GPR.write(rt, 0xFFFF & MEM.read_half(end));
  GPR.write(ra,end);
  imprime("Resultado: r%d <-- %d -- r%d <-- %d\n",rt, GPR.read(rt), ra, GPR.read(ra));
}
//------------------------------------------------------------------------------
void ac_behavior(lhzx)
{
  printInstruction("lhzx %d, %d, %d\n", rt, ra, rb);
  unsigned end = ra==0 ? GPR.read(rb) : GPR.read(ra) + GPR.read(rb);
  GPR.write(rt, 0xFFFF & MEM.read_half(end));
  imprime("Resultado: r%d <-- %d\n",rt, GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior(lmw)
{
  printInstruction("lmw %d, %d(%d)\n", rt, imm, ra);
  unsigned end = ra==0 ? imm : GPR.read(ra) + imm;
  unsigned r = rt;
  while(r <32) {
    GPR.write(r, MEM.read(end));
    r++;
    end += 4;
  }

  // DSI Exception ?? (PENDENTE)
  printf("TESTE");
}
//------------------------------------------------------------------------------
void ac_behavior(lswi)
{
  printInstruction("lswi %d, %d, %d\n", rt, ra, rb);
```

```
  unsigned end = ra==0 ? 0 : GPR.read(ra);
  unsigned n = rb==0 ? 32 : GPR.read(rb);
  int r = rt -1;
  int i=0;
  unsigned valor = 0;
  while (n>0) {
    if (i==0){
      r = (r + 1)%32;
      GPR.write(r,0);
    }
    unsigned temp = MEM.read_byte(end) << (24 - i);
    valor = valor | temp;
    i = i+8;
    if(i==32) {
      i = 0;
      valor = 0;
      GPR.write(r, valor);
    }
    end++;
    n--;
  }
  // DSI Exception ?? (PENDENTE)
  printf("TESTE");
}
//------------------------------------------------------------------------------
void ac_behavior(lswx)
{
  printInstruction("lswx %d, %d, %d\n", rt, ra, rb);
  if (rt==ra || rt == rb || (rt==0 && rb==0)){
    imprime_string("instrucao invalida");
    return;
  }

  unsigned end = ra==0 ? 0 : GPR.read(ra);
  unsigned n = XER.bc;
  int r = rt -1;
  int i=0;
  unsigned valor = 0;
  while (n>0) {
    if (i==0){
      r = (r + 1)%32;
      GPR.write(r,0);
    }
    unsigned temp = MEM.read_byte(end) << (24 - i);
    valor = valor | temp;
    i = i+8;
    if(i==32) {
      i = 0;
      valor = 0;
```

```
      GPR.write(r, valor);
    }
    end++;
    n--;
  }
  // DSI Exception ?? (PENDENTE)
  printf("TESTE");
}
//------------------------------------------------------------------------------
void ac_behavior(lwarx)
{
  printInstruction("lwarx %d, %d, %d\n", rt, ra, rb);
  printf("\nNao implementado\n");
  // PENDENTE
}
//------------------------------------------------------------------------------
void ac_behavior(lwbrx)
{
  imprime("lwbrx %d, %d, %d\n", rt, ra, rb);
  unsigned end = ra==0 ? GPR.read(rb) : GPR.read(ra) + GPR.read(rb);
  unsigned temp = MEM.read_byte(end+3);
  unsigned valor = temp << 24;
  temp = MEM.read_byte(end+2);
  valor = valor | temp << 16;
  temp = MEM.read_byte(end+1);
  valor = valor | temp << 8;
  temp = MEM.read_byte(end+1);
  valor = valor | temp;
  GPR.write(rt, valor);
  imprime("Resultado: r%d <-- %d\n",rt, GPR.read(rt));
  printf("TESTE");
}
//------------------------------------------------------------------------------
void ac_behavior(lwzu)
{
  printInstruction("lzu %d, %d(%d)\n", rt, imm, ra);
  if (ra==0 || ra==rt){
    imprime_string("instrucao invalida");
    return;
  }
  unsigned end = GPR.read(ra) + imm;
  GPR.write(rt, MEM.read(end));
  GPR.write(ra, end);
  imprime("Resultado: r%d <-- %d -- r%d <-- %d\n",rt, GPR.read(rt), ra, GPR.read(ra));
}
//------------------------------------------------------------------------------
void ac_behavior(lwzux)
{
  printInstruction("lwzux %d, %d, %d\n", rt, ra, rb);
```

```
  if (ra==0 || ra==rt){
    imprime_string("instrucao invalida");
    return;
  }
  unsigned end = GPR.read(ra) + GPR.read(rb);
  GPR.write(rt, MEM.read(end));
  GPR.write(ra, end);
  imprime("Resultado: r%d <-- %d -- r%d <-- %d\n",rt, GPR.read(rt), ra, GPR.read(ra));

}
//-------------------------------------------------------------------------
void ac_behavior(lwzx)
{
  printInstruction("lwzx %d, %d, %d\n", rt, ra, rb);
  unsigned end = ra==0 ? GPR.read(rb) : GPR.read(ra) + GPR.read(rb);
  GPR.write(rt, MEM.read(end));
  imprime("Resultado: r%d <-- %d\n",rt, GPR.read(rt));

}
//-------------------------------------------------------------------------
void ac_behavior(addic)
{
  printInstruction("addic %d, %d, %d\n", rt, ra, imm);
  unsigned immU32 = imm;
  XER.ca = CarryOut((UINT64)GPR.read(ra) + immU32);
  GPR.write(rt, GPR.read(ra) + imm);
  imprime("Resultado ->  rt = %d\n\n", GPR.read(rt));
}
//-------------------------------------------------------------------------
void ac_behavior(addic_dot)
{
  printInstruction("addic. %d, %d, %d\n", rt, ra, imm);
  unsigned immU32 = imm;
  XER.ca = CarryOut((UINT64)GPR.read(ra) + immU32);
  GPR.write(rt, GPR.read(ra) + imm);
  atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  rt = %d\n\n", GPR.read(rt));
}
//-------------------------------------------------------------------------
void ac_behavior(addme)
{
  printInstruction("addme %d, %d\n",rt, ra);
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut(GPR.read(ra) + 0xFFFFFFFF+ OldXERca);
  GPR.write(rt, GPR.read(ra) + OldXERca - 1);
  imprime("Resultado ->  rt = %d\n\n", GPR.read(rt));
}
//-------------------------------------------------------------------------
void ac_behavior(addme_dot)
```

```
{
  printInstruction("addme. %d, %d\n",rt, ra);
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut(GPR.read(ra) + 0xFFFFFFFF+ OldXERca);
  GPR.write(rt, GPR.read(ra) + OldXERca - 1);
  atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  rt = %d\n\n", GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(addmeo)
{
  printInstruction("addmeo %d, %d\n",rt, ra);
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut(GPR.read(ra) + 0xFFFFFFFF+ OldXERca);
  GPR.write(rt, GPR.read(ra) + OldXERca - 1);
  SetXERAddOverflow(GPR.read(ra), OldXERca - 1, GPR.read(rt));
  imprime("Resultado ->  rt = %d\n\n", GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(addmeo_dot)
{
  printInstruction("addmeo. %d, %d\n",rt, ra);
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut(GPR.read(ra) + 0xFFFFFFFF+ OldXERca);
  GPR.write(rt, GPR.read(ra) + OldXERca - 1);
  atualizaCR0(GPR.read(rt));
  SetXERAddOverflow(GPR.read(ra), OldXERca - 1, GPR.read(rt));
  imprime("Resultado ->  rt = %d\n\n", GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(addze)
{
  printInstruction("addze %d, %d\n",rt, ra);
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut(GPR.read(ra) + OldXERca);
  GPR.write(rt, GPR.read(ra) + OldXERca);
  imprime("Resultado ->  rt = %d\n\n", GPR.read(rt));

}
//------------------------------------------------------------------------
void ac_behavior(addze_dot)
{
  printInstruction("addze. %d, %d\n",rt, ra);
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut(GPR.read(ra) + OldXERca);
  GPR.write(rt, GPR.read(ra) + OldXERca);
  atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  rt = %d\n\n", GPR.read(rt));
```

```
}
//------------------------------------------------------------------------------
void ac_behavior(addzeo)
{
  printInstruction("addzeo %d, %d\n",rt, ra);
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut(GPR.read(ra) + OldXERca);
  GPR.write(rt, GPR.read(ra) + OldXERca);
  SetXERAddOverflow(GPR.read(ra), OldXERca , GPR.read(rt));
  imprime("Resultado ->  rt = %d\n\n", GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior(addzeo_dot)
{
  printInstruction("addzeo. %d, %d\n",rt, ra);
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut(GPR.read(ra) + OldXERca);
  GPR.write(rt, GPR.read(ra) + OldXERca);
  atualizaCR0(GPR.read(rt));
  SetXERAddOverflow(GPR.read(ra), OldXERca , GPR.read(rt));
  imprime("Resultado ->  rt = %d\n\n", GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior(bcctr)
{
  printInstruction("bcctr %d, %d\n",rt, ra);
  unsigned cr = (CR >>(31 - ra)) & 0x1;
  unsigned bo0 = (rt & 0x10)>>4;
  unsigned bo1 = (rt & 0x8)>>3;
  unsigned bo2 = (rt & 0x4)>>2;
  unsigned bo3 = (rt & 0x2)>>1;

  if(bo2 == 0)
    CTR = CTR - 1;
  if((bo2==1 || ((CTR==0) == bo3)) && (bo0 ==1 || (cr == bo1)))
  {
    ac_pc = CTR & 0xFFFFFFFC;
  }

  imprime("Resultado ->  ac_pc <- %d\n", (int)ac_pc);
}
//------------------------------------------------------------------------------
void ac_behavior(bcctrl)
{
  printInstruction("bcctrl %d, %d\n",rt, ra);
  unsigned cr = (CR >>(31 - ra)) & 0x1;
  unsigned bo0 = (rt & 0x10)>>4;
  unsigned bo1 = (rt & 0x8)>>3;
  unsigned bo2 = (rt & 0x4)>>2;
```

```
  unsigned bo3 = (rt & 0x2)>>1;

  LK = ac_pc;
  if(bo2 == 0)
    CTR = CTR - 1;
  if((bo2==1 || ((CTR==0) == bo3)) && (bo0 ==1 || (cr == bo1)))
    ac_pc = CTR & 0xFFFFFFFC;

  imprime("Resultado ->  ac_pc <- %d\n", (int)ac_pc);
}
//------------------------------------------------------------------------------
void ac_behavior(bclr)
{
  printInstruction("bclr %d, %d\n",rt, ra);
  unsigned cr = (CR >>(31 - ra)) & 0x1;
  unsigned bo0 = (rt & 0x10)>>4;
  unsigned bo1 = (rt & 0x8)>>3;
  unsigned bo2 = (rt & 0x4)>>2;
  unsigned bo3 = (rt & 0x2)>>1;

  if(bo2 == 0)
    CTR = CTR - 1;
  if((bo2==1 || ((CTR==0) == bo3)) && (bo0 ==1 || (cr == bo1)))
    ac_pc = LK & 0xFFFFFFFC;

  imprime("Resultado ->  ac_pc <- %d\n", (int)ac_pc);
}
//------------------------------------------------------------------------------
void ac_behavior(bclrl)
{
  printInstruction("bclrl %d, %d\n",rt, ra);
  unsigned cr = (CR >>(31 - ra)) & 0x1;
  unsigned bo0 = (rt & 0x10)>>4;
  unsigned bo1 = (rt & 0x8)>>3;
  unsigned bo2 = (rt & 0x4)>>2;
  unsigned bo3 = (rt & 0x2)>>1;

  unsigned nia = ac_pc;
  if(bo2 == 0)
    CTR = CTR - 1;
  if((bo2==1 || ((CTR==0) == bo3)) && (bo0 ==1 || (cr == bo1)))
    ac_pc = LK & 0xFFFFFFFC;

  LK = nia;

  imprime("Resultado ->  ac_pc <- %d\n", (int)ac_pc);
}
//------------------------------------------------------------------------------
void ac_behavior(cntlzw)
```

```
{
  printInstruction("cntlzw %d, %d\n",ra, rt);
  unsigned mask = 0x80000000;
  unsigned n = 0;
  while(n<32){
    if (rt & mask)
      break;
    n++;
    mask = mask >>1;
  }
  GPR.write(ra,n);
  imprime("Resultado ->  r%d <- %d\n", ra, GPR.read(ra));
}
//-----------------------------------------------------------------------------
void ac_behavior(cntlzw_dot)
{
  printInstruction("cntlzw. %d, %d\n",ra, rt);
  unsigned mask = 0x80000000;
  unsigned n = 0;
  while(n<32){
    if (rt & mask)
      break;
    n++;
    mask = mask >>1;
  }
  GPR.write(ra,n);
  atualizaCR0(GPR.read(ra));
  imprime("Resultado ->  r%d <- %d\n", ra, GPR.read(ra));
}
//-----------------------------------------------------------------------------
void ac_behavior(divw)
{
  printInstruction("divw %d, %d, %d\n",rt, ra, rb);
  int dividendo = GPR.read(ra);
  int divisor = GPR.read(rb);
  if (divisor == 0 || ((unsigned)dividendo == 0x80000000 && divisor == -1))
    return;
  GPR.write(rt, dividendo / divisor);
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//-----------------------------------------------------------------------------
void ac_behavior(divw_dot)
{
  printInstruction("divw. %d, %d, %d\n",rt, ra, rb);
  int dividendo = GPR.read(ra);
  int divisor = GPR.read(rb);
  if (divisor == 0 || ((unsigned)dividendo == 0x80000000 && divisor == -1))
    return;
  GPR.write(rt, dividendo / divisor);
```

```
  atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(divwo)
{
  printInstruction("divwo %d, %d, %d\n",rt, ra, rb);
  int dividendo = GPR.read(ra);
  int divisor = GPR.read(rb);
  if (divisor == 0 || ((unsigned)dividendo == 0x80000000 && divisor == -1)){
    XER.ov = 1;
    return;
  }
  GPR.write(rt, dividendo / divisor);
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(divwo_dot)
{
  printInstruction("divwo. %d, %d, %d\n",rt, ra, rb);
  int dividendo = GPR.read(ra);
  int divisor = GPR.read(rb);
  if (divisor == 0 || ((unsigned)dividendo == 0x80000000 && divisor == -1)){
    XER.ov = 1;
    return;
  }
  GPR.write(rt, dividendo / divisor);
  atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(divwu)
{
  printInstruction("divwu %d, %d, %d\n",rt, ra, rb);
  unsigned dividendo = GPR.read(ra);
  unsigned divisor = GPR.read(rb);
  if (divisor == 0)
    printf("ERRO!!");
//    return;
  GPR.write(rt, dividendo / divisor);
  imprime("Resultado ->  r%d <- %d\n", rt, (unsigned)GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(divwu_dot)
{
  printInstruction("divuw. %d, %d, %d\n",rt, ra, rb);
  unsigned dividendo = GPR.read(ra);
  unsigned divisor = GPR.read(rb);
  if (divisor == 0)
```

```
    return;
  GPR.write(rt, dividendo / divisor);
  atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, (unsigned)GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(divwuo)
{
  printInstruction("divwuo %d, %d, %d\n",rt, ra, rb);
  unsigned dividendo = GPR.read(ra);
  unsigned divisor = GPR.read(rb);
  if (divisor == 0){
    XER.ov = 1;
    return;
  }
  GPR.write(rt, dividendo / divisor);
  imprime("Resultado ->  r%d <- %d\n", rt, (unsigned)GPR.read(rt));

}
//------------------------------------------------------------------------
void ac_behavior(divwuo_dot)
{
  printInstruction("divwuo. %d, %d, %d\n",rt, ra, rb);
  unsigned dividendo = GPR.read(ra);
  unsigned divisor = GPR.read(rb);
  if (divisor == 0){
    XER.ov = 1;
  XER.so = 1;
    printf("ERRO");
  return;
  }
  GPR.write(rt, dividendo / divisor);
  atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, (unsigned)GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(extsb)
{
  printInstruction("extsb %d, %d\n",ra, rt);
  char temp = 0xFF & GPR.read(rt);
  GPR.write(ra, (int)temp);
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior(extsb_dot)
{
  printInstruction("extsb. %d, %d\n",ra, rt);
  char temp = 0xFF & GPR.read(rt);
  GPR.write(ra, (int)temp);
```

```
  atualizaCR0(GPR.read(ra));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior(extsh)
{
  printInstruction("extsh %d, %d\n",ra, rt);
  short int temp = 0xFFFF & GPR.read(rt);
  GPR.write(ra, (int)temp);
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior(extsh_dot)
{
  printInstruction("extsh. %d, %d\n",ra, rt);
  short int temp = 0xFFFF & GPR.read(rt);
  GPR.write(ra, (int)temp);
  atualizaCR0(GPR.read(ra));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(ra));

}
//------------------------------------------------------------------------
void ac_behavior(mulhw)
{
  printInstruction("mulhw %d, %d, %d\n",rt, ra, rb);
  long long resultado64 = GPR.read(ra);
  resultado64 *= GPR.read(rb);
  int resultado32 = ((resultado64 >> 32) & 0xFFFFFFFF);
  GPR.write(rt, resultado32);
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(mulhw_dot)
{
  printInstruction("mulhw. %d, %d, %d\n",rt, ra, rb);
  long long resultado64 = GPR.read(ra);
  resultado64 *= GPR.read(rb);
  int resultado32 = ((resultado64 >> 32) & 0xFFFFFFFF);
  GPR.write(rt, resultado32);
  atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(mulhwu)
{
  printInstruction("mulhwu %d, %d, %d\n",rt, ra, rb);
  unsigned long long resultado64 = GPR.read(ra);
  resultado64 *= GPR.read(rb);
  unsigned  int resultado32 = ((resultado64 >> 32) & 0xFFFFFFFF);
```

```
  GPR.write(rt, resultado32);
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------

void ac_behavior(mulhwu_dot)
{
  printInstruction("mulhwu. %d, %d, %d\n",rt, ra, rb);
  unsigned long long resultado64 = GPR.read(ra);
  resultado64 *= GPR.read(rb);
  unsigned  int resultado32 = ((resultado64 >> 32) & 0xFFFFFFFF);
  GPR.write(rt, resultado32);
  atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}

//------------------------------------------------------------------------
void ac_behavior(mulli)
{
  printInstruction("mulli %d, %d, %d\n",rt, ra, imm);
  long long resultado64 = GPR.read(ra);
  resultado64 *= imm;
  int resultado32 = resultado64 & 0xFFFFFFFF;
  GPR.write(rt, resultado32);
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));

}
//------------------------------------------------------------------------
void ac_behavior(mullw)
{
  printInstruction("mullw %d, %d, %d\n",rt, ra, rb);
  long long resultado64 = GPR.read(ra);
  resultado64 *= GPR.read(rb);
  int resultado32 = resultado64 & 0xFFFFFFFF;
  GPR.write(rt, resultado32);
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(mullw_dot)
{
  printInstruction("mullw. %d, %d, %d\n",rt, ra, rb);
  long long resultado64 = GPR.read(ra);
  resultado64 *= GPR.read(rb);
  int resultado32 = resultado64 & 0xFFFFFFFF;
  GPR.write(rt, resultado32);
  atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));

}
```

```
//------------------------------------------------------------------------
void ac_behavior(mullwo)
{
  printInstruction("mullwo %d, %d, %d\n",rt, ra, rb);
  long long resultado64 = GPR.read(ra);
  resultado64 *= GPR.read(rb);
  int resultado32 = resultado64 & 0xFFFFFFFF;
  XER.ov = resultado64 > 0x7fffffff || resultado64 < 0x80000000;
  GPR.write(rt, resultado32);
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(mullwo_dot)
{
  printInstruction("mullwo. %d, %d, %d\n",rt, ra, rb);
  long long resultado64 = GPR.read(ra);
  resultado64 *= GPR.read(rb);
  int resultado32 = resultado64 & 0xFFFFFFFF;
  XER.ov = resultado64 > 0x7fffffff || resultado64 < 0x80000000;
  GPR.write(rt, resultado32);
  atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));

}
//------------------------------------------------------------------------
void ac_behavior(slw)
{
  printInstruction("slw %d, %d, %d\n",ra, rt, rb);
  unsigned m;
  unsigned n = GPR.read(rb) & 0x1F;
  unsigned r = rotl(GPR.read(rt), n);
  if (rb&0x20)
    m = 0x0;
  else
    m = mask(0,31-n);
  GPR.write(ra, r & m);

  imprime("Resultado ->  r%d <- %d\n", ra, GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior(slw_dot)
{
  printInstruction("slw. %d, %d, %d\n",ra, rt, rb);
  unsigned m;
  unsigned n = GPR.read(rb) & 0x1F;
  unsigned r = rotl(GPR.read(rt), n);
  if (rb&0x20)
    m = 0x0;
  else
```

```
    m = mask(0,31-n);
  GPR.write(ra, r & m);
  atualizaCR0(GPR.read(ra));
  imprime("Resultado ->  r%d <- %d\n", ra, GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior(sraw)
{
  printInstruction("sraw %d, %d, %d\n",ra, rt, rb);
  unsigned m;
  unsigned n = GPR.read(rb) & 0x1F;
  unsigned r = rotl(GPR.read(rt), 32-n);
  if (rb&0x20)
    m = 0x0;
  else
    m = mask(n,31);
  unsigned s;
  if(GPR.read(rt)&0x80000000)
    s = 0xFFFFFFFF;
  else
    s = 0x00000000;
  unsigned result = (r & m) | (s & ~m);

  GPR.write(ra,result);

  XER.ca = s & ((r & !m) != 0);
  printf("TESTE sraw\n");
}
//------------------------------------------------------------------------
void ac_behavior(sraw_dot)
{
  printInstruction("sraw. %d, %d, %d\n",ra, rt, rb);
  unsigned m;
  unsigned n = GPR.read(rb) & 0x1F;
  unsigned r = rotl(GPR.read(rt), 32-n);
  if (rb&0x20)
    m = 0x0;
  else
    m = mask(n,31);
  unsigned s;
  if(GPR.read(rt)&0x80000000)
    s = 0xFFFFFFFF;
  else
    s = 0x00000000;
  unsigned result = (r & m) | (s & ~m);

  GPR.write(ra,result);
  XER.ca = s & ((r & !m) != 0);
  printf("TESTE sraw.\n");
```

```
  atualizaCR0(GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior(srawi)
{
  printInstruction("srawi %d, %d, %d\n",ra, rt, rb);
  int temp = GPR.read(rt);
  temp = temp >> rb;
  GPR.write(ra, temp);
  XER.ca = (rb != 0) && ((int) GPR.read(ra) < 0) && ((GPR.read(rt) << (32 - rb)) != 0);

/*
  unsigned n = rb;
  unsigned r = rotl(GPR.read(rt), 32 - n);
  unsigned m = mask(n, 31);
  unsigned s = (GPR.read(rt) & 0x80000000)>>31;
  unsigned s32;
  if(s)
    s32 = 0xFFFFFFFF;
  else
    s32 = 0x00000000;
  unsigned result = (r & m) | (s32 & ~m);
  GPR.write(ra, result);
  printf("TESTE srawi\n");
  XER.ca = s & ((r & ~m) != 0);
  */
}
//------------------------------------------------------------------------
void ac_behavior(srawi_dot)
{
  printInstruction("srawi. %d, %d, %d\n",ra, rt, rb);
  int temp = GPR.read(rt);
  temp = temp >> rb;
  GPR.write(ra, temp);
  XER.ca = (rb != 0) && ((int) GPR.read(ra) < 0) && ((GPR.read(rt) << (32 - rb)) != 0);

/*
  unsigned n = rb;
  unsigned r = rotl(GPR.read(rt), 32 - n);
  unsigned m = mask(n, 31);
  unsigned s = (GPR.read(rt) & 0x80000000)>>31;
  unsigned s32;
  if(s)
    s32 = 0xFFFFFFFF;
  else
    s32 = 0x00000000;
  unsigned result = (r & m) | (s32 & ~m);
  GPR.write(ra, result);
  XER.ca = s & ((r & ~m) != 0);
```

```
  printf("TESTE srawi.\n");
  */
  atualizaCR0(GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior(srw)
{
  printInstruction("srw %d, %d, %d\n",ra, rt, rb);
  unsigned m;
  unsigned n = GPR.read(rb)&0x1f;
  unsigned r = rotl(GPR.read(rt), 32-n);
  if (rb&0x20)
    m = 0x0;
  else
    m = mask(n,31);
  GPR.write(ra, r & m);

  imprime("Resultado ->  r%d <- %d\n", ra, GPR.read(ra));
}
//------------------------------------------------------------------------
void ac_behavior(srw_dot)
{
  printInstruction("srw. %d, %d, %d\n",ra, rt, rb);
  unsigned m;
  unsigned n = GPR.read(rb)&0x1f;
  unsigned r = rotl(GPR.read(rt), 32-n);
  if (rb&0x20)
    m = 0x0;
  else
    m = mask(n,31);
  GPR.write(ra, r & m);

  atualizaCR0(GPR.read(ra));
  imprime("Resultado ->  r%d <- %d\n", ra, GPR.read(ra));
}
//------------------------------------------------------------------------

void ac_behavior(rlwimi)
{
  printInstruction("rlwimi %d, %d, %d, %d, %d\n",ra, rt, rb, mb, me);
  unsigned r = rotl(GPR.read(rt), rb);
  unsigned m = mask(mb, me);
  GPR.write(ra, (r & m) | (GPR.read(ra) & ~m));
  imprime("Resultado ->  r%d <- %d\n", ra, GPR.read(ra));
}

//------------------------------------------------------------------------
void ac_behavior(rlwimi_dot)
{
```

```
  printInstruction("rlwimi. %d, %d, %d, %d, %d\n",ra, rt, rb, mb, me);
  unsigned r = rotl(GPR.read(rt), rb);
  unsigned m = mask(mb, me);
  GPR.write(ra, (r & m) | (GPR.read(ra) & ~m));
  atualizaCR0(GPR.read(ra));
  imprime("Resultado ->  r%d <- %d\n", ra, GPR.read(ra));

}
//------------------------------------------------------------------------------
void ac_behavior(rlwinm)
{
  printInstruction("rlwinm %d, %d, %d, %d, %d\n",ra, rt, rb, mb, me);
  unsigned r = rotl(GPR.read(rt), rb);
  unsigned m = mask(mb, me);
  GPR.write(ra, (r & m));
  imprime("Resultado ->  r%d <- %d\n", ra, GPR.read(ra));
}
//------------------------------------------------------------------------------
void ac_behavior(rlwinm_dot)
{
  printInstruction("rlwinm. %d, %d, %d, %d, %d\n",ra, rt, rb, mb, me);
  unsigned r = rotl(GPR.read(rt), rb);
  unsigned m = mask(mb, me);
  GPR.write(ra, (r & m));
  atualizaCR0(GPR.read(ra));
  imprime("Resultado ->  r%d <- %d\n", ra, GPR.read(ra));

}
//------------------------------------------------------------------------------
void ac_behavior(rlwnm)
{
  printInstruction("rlwnm %d, %d, %d, %d, %d\n",ra, rt, rb, mb, me);
  unsigned n = GPR.read(rb) & 0x1F;
  unsigned r = rotl(GPR.read(rt), n);
  unsigned m = mask(mb, me);
  GPR.write(ra, (r & m));
  imprime("Resultado ->  r%d <- %d\n", ra, GPR.read(ra));

}
//------------------------------------------------------------------------------
void ac_behavior(rlwnm_dot)
{
  printInstruction("rlwnm. %d, %d, %d, %d, %d\n",ra, rt, rb, mb, me);
  unsigned n = GPR.read(rb) & 0x1F;
  unsigned r = rotl(GPR.read(rt), n);
  unsigned m = mask(mb, me);
  GPR.write(ra, (r & m));
  atualizaCR0(GPR.read(ra));
  imprime("Resultado ->  r%d <- %d\n", ra, GPR.read(ra));
```

```
}
//------------------------------------------------------------------------------
void ac_behavior(subfc)
{
  printInstruction("subfc %d, %d, %d\n",rt, ra, rb);
  XER.ca = CarryOut((UINT64)~GPR.read(ra) + GPR.read(rb)+1);
  GPR.write(rt, GPR.read(rb) - GPR.read(ra));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior(subfc_dot)
{
  printInstruction("subfc. %d, %d, %d\n",rt, ra, rb);
  XER.ca = CarryOut((UINT64)~GPR.read(ra) +GPR.read(rb)+1);
  GPR.write(rt, GPR.read(rb) - GPR.read(ra));
  atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior(subfco)
{
  printInstruction("subfco %d, %d, %d\n",rt, ra, rb);
  XER.ca = CarryOut((UINT64)~GPR.read(ra) + GPR.read(rb)+1);
  GPR.write(rt, GPR.read(rb) - GPR.read(ra));
  SetXERSubOverflow(GPR.read(ra), GPR.read(rb), GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior(subfco_dot)
{
  printInstruction("subfco. %d, %d, %d\n",rt, ra, rb);
  XER.ca = ((UINT64)~GPR.read(ra) + GPR.read(rb)+1);
  GPR.write(rt, GPR.read(rb) - GPR.read(ra));
  SetXERSubOverflow(GPR.read(ra), GPR.read(rb), GPR.read(rt));
  atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------------
void ac_behavior(subfe)
{
  printInstruction("subfe %d, %d, %d\n",rt, ra, rb);
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut((UINT64)~GPR.read(ra) + GPR.read(rb)+OldXERca);
  GPR.write(rt, ~GPR.read(ra) + GPR.read(rb) + OldXERca);
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));

}
//------------------------------------------------------------------------------
void ac_behavior(subfe_dot)
```

```
{
  printInstruction("subfe. %d, %d, %d\n",rt, ra, rb);
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut((UINT64)~GPR.read(ra) + GPR.read(rb)+OldXERca);
  GPR.write(rt, ~GPR.read(ra) + GPR.read(rb) + OldXERca);
  atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//-----------------------------------------------------------------------------
void ac_behavior(subfeo)
{
  printInstruction("subfeo %d, %d, %d\n",rt, ra, rb);
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut((UINT64)~GPR.read(ra) + GPR.read(rb)+OldXERca);
  GPR.write(rt, ~GPR.read(ra) + GPR.read(rb) + OldXERca);
  SetXERAddOverflow(~GPR.read(ra), GPR.read(rb) + OldXERca, GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//-----------------------------------------------------------------------------
void ac_behavior(subfeo_dot)
{
  printInstruction("subfeo. %d, %d, %d\n",rt, ra, rb);
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut((UINT64)~GPR.read(ra) + GPR.read(rb)+OldXERca);
  GPR.write(rt, ~GPR.read(ra) + GPR.read(rb) + OldXERca);
  atualizaCR0(GPR.read(rt));
  SetXERAddOverflow(~GPR.read(ra), GPR.read(rb) + OldXERca, GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//-----------------------------------------------------------------------------
void ac_behavior(subfic)
{
  printInstruction("subfic %d, %d, %d\n",rt, ra, imm);
  unsigned immU32 = imm;
  XER.ca = CarryOut((UINT64)~GPR.read(ra) + immU32 +1);
  GPR.write(rt, imm - GPR.read(ra));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//-----------------------------------------------------------------------------
inline void generic_subfme(unsigned rt, unsigned ra, unsigned rb, bool oe, bool rc)
{
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut((UINT64)~GPR.read(ra)+ OldXERca + 0xFFFFFFFF);
  GPR.write(rt, ~GPR.read(ra)+OldXERca-1);
  if(oe)
    SetXERAddOverflow(~GPR.read(ra), OldXERca-1, GPR.read(rt));
  if(rc)
    atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
```

```
}
//-------------------------------------------------------------------------
void ac_behavior(subfme)
{
  printInstruction("subfme %d, %d, %d\n",rt, ra, rb);
  generic_subfme(rt, ra, rb,false,false);
}
//-------------------------------------------------------------------------
void ac_behavior(subfme_dot)
{
  printInstruction("subfme. %d, %d, %d\n",rt, ra, rb);
  generic_subfme(rt, ra, rb,false,true);
}
//-------------------------------------------------------------------------
void ac_behavior(subfmeo)
{
  printInstruction("subfmeo %d, %d, %d\n",rt, ra, rb);
  generic_subfme(rt, ra, rb,true,false);
}
//-------------------------------------------------------------------------
void ac_behavior(subfmeo_dot)
{
  printInstruction("subfmeo. %d, %d, %d\n",rt, ra, rb);
  generic_subfme(rt, ra, rb,true,true);
}
//-------------------------------------------------------------------------
inline void generic_subfze(unsigned rt, unsigned ra, unsigned rb, bool oe, bool rc)
{
  unsigned OldXERca = XER.ca;
  XER.ca = CarryOut((UINT64)~GPR.read(ra) + OldXERca);
  GPR.write(rt, ~GPR.read(ra)+OldXERca);
  if(oe)
    SetXERAddOverflow(~GPR.read(ra), OldXERca, GPR.read(rt));
  if(rc)
    atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//-------------------------------------------------------------------------
void ac_behavior(subfze)
{
  printInstruction("subfze %d, %d, %d\n",rt, ra, rb);
  generic_subfze(rt, ra, rb,false, false);
}
//-------------------------------------------------------------------------
void ac_behavior(subfze_dot)
{
  printInstruction("subfze. %d, %d, %d\n",rt, ra, rb);
  generic_subfze(rt, ra, rb,false, true);
}
```

```
//------------------------------------------------------------------------
void ac_behavior(subfzeo)
{
  printInstruction("subfzeo %d, %d, %d\n",rt, ra, rb);
  generic_subfze(rt, ra, rb,true, false);
}
//------------------------------------------------------------------------
void ac_behavior(subfzeo_dot)
{
  printInstruction("subfzeo. %d, %d, %d\n",rt, ra, rb);
  generic_subfze(rt, ra, rb,true, true);
}
//------------------------------------------------------------------------
inline void generic_subf(unsigned rt, unsigned ra, unsigned rb, bool oe, bool rc)
{
  GPR.write(rt, GPR.read(rb)-GPR.read(ra));
  if(oe)
    SetXERSubOverflow(GPR.read(rb), GPR.read(ra), GPR.read(rt));
  if(rc)
    atualizaCR0(GPR.read(rt));
  imprime("Resultado ->  r%d <- %d\n", rt, GPR.read(rt));
}
//------------------------------------------------------------------------
void ac_behavior(subf_dot)
{
  printInstruction("subf. %d, %d, %d\n",rt, ra, rb);
  generic_subf(rt, ra, rb,false,true);
}
//------------------------------------------------------------------------
void ac_behavior(subfo)
{
  printInstruction("subfo %d, %d, %d\n",rt, ra, rb);
  generic_subf(rt, ra, rb,true,false);
}
//------------------------------------------------------------------------
void ac_behavior(subfo_dot)
{
  printInstruction("subfo. %d, %d, %d\n",rt, ra, rb);
  generic_subf(rt, ra, rb,true,true);
}
//------------------------------------------------------------------------

void ac_behavior(cmpi)
{
  unsigned bf = (rt & 0x1C) >> 2;
  printInstruction("cmpi %d, 0, %d, %d\n", bf, ra, imm);
  GenericCompare(bf, GPR.read(ra),imm);
}
//------------------------------------------------------------------------
```

```
void ac_behavior(cmpl)
{
  unsigned bf = (rt & 0x1C) >> 2;
  printInstruction("cmpl %d, 0, %d, %d\n", bf, ra, rb);
  GenericLogicCompare(bf, GPR.read(ra),GPR.read(rb));

}
//------------------------------------------------------------------------------
void ac_behavior(cmpli)
{
  unsigned bf = (rt & 0x1C) >> 2;
  printInstruction("cmpli %d, 0, %d, %d\n", bf, ra, imm);
  unsigned immField = 0xFFFF & imm;
  GenericLogicCompare(bf, GPR.read(ra),immField);
}
//------------------------------------------------------------------------------
void ac_behavior(mcrf)
{
  printInstruction("mcrf %d, %d\n", crfD, crfS);
  printf("mcrf  - Nao implementado!!");
}
//------------------------------------------------------------------------------
void ac_behavior(mcrxr)
{
  printInstruction("mcrxr %d\n", crfD);
  printf("mcrxr  - Nao implementado!!");
}
//------------------------------------------------------------------------------
void ac_behavior(mfcr)
{
  printInstruction("mfcr %d\n", rt);
  GPR.write(rt, CR);
}
//------------------------------------------------------------------------------
void ac_behavior(mfmsr)
{
  printInstruction("mfmsr %d\n", rt);
  printf("mfmsr  - Nao implementado!!");
}
//------------------------------------------------------------------------------
void ac_behavior(mtmsr)
{
  printInstruction("mtmsr %d\n", rt);
  printf("mtmsr  - Nao implementado!!");
}
//------------------------------------------------------------------------------
void ac_behavior(rfi)
{
  imprime_string("rfi");
```

```
  printf("rfi  - Nao implementado!!");
}
//------------------------------------------------------------------------
void ac_behavior(sc)
{
// Não implementado!! Esta sendo usado como debug, mostrando o endereco 0x00
  imprime_string("sc");
//  printf("\nsc  - Nao implementado!!
        Esta sendo usado como debug, mostrando o endereco 0x00\n");
  printf("%d\n", MEM.read(0));
}
//------------------------------------------------------------------------
void ac_behavior(stwcx_dot)
{
  printInstruction("stwcx. %d, %d, %d\n", rt, ra, rb);
  printf("stwcx.  - Nao implementado!!");
}
//------------------------------------------------------------------------
void ac_behavior(mfspr)
{
  printInstruction("mfspr %d, %d\n",rt, spr);
  // Pendente - Implementar outros registradores
  unsigned numero = (spr & 0x1F) << 5;
  numero = numero | ((spr & 0x3E0) >> 5);
  switch(numero){
//    case 0x01:
//      XER = rt;
//    break;
    case 0x08:
      GPR.write(rt, LK);
    break;
    case 0x09:
     GPR.write(rt, CTR);
    break;
    default:
      printf("mfspr com registrador = %d - Não implementado ainda...",numero);
    break;
  }
}
//------------------------------------------------------------------------
void ac_behavior(mftb)
{
  printInstruction("mftb %d, %d\n",rt, spr);
  printf("mftb - Nao implementado!!");
}
//------------------------------------------------------------------------
void ac_behavior(mtspr)
{
  printInstruction("mtspr %d, %d\n",rt, spr);
```

```
  // Pendente - Implementar outros registradores
  unsigned numero = (spr & 0x1F) << 5;
  numero = numero | ((spr & 0x3E0) >> 5);
  switch(numero){
//    case 0x01:
//      XER = rt;
//    break;
    case 0x08:
      LK = GPR.read(rt);
    break;
    case 0x09:
      CTR = GPR.read(rt);
    break;
    default:
      printf("mtspr com registrador = %d - Não implementado ainda...",numero);
    break;
  }
}
//------------------------------------------------------------------------------
void ac_behavior(mtcrf)
{
  unsigned fxm = (spr >> 1) & 0xFF;
  printInstruction("mtcrf %d, %d\n",rt,  fxm);
  unsigned mask = 0x0;
  for(int i = 0; i<8; i++){
  unsigned bit = (fxm << i) & 0x80;
  if(bit)
      mask = mask | (0xF0000000 >> (i*4));
  }
  CR = ((GPR.read(rt)&mask) | (CR & ~mask));
  imprime("Resultado = CR=%d\n",(int)CR);
}

//------------------------------------------------------------------------------
```

5.2 Anexo II - Fontes do Gerador de Montadores

5.2.1 Arquivo main.cpp

```
#include "Main.h"
#include <fstream>
#include <iostream>
#include <string>

Main::Main(string nomeArquivoArchC, string nomeArquivoGALS)
{
  bool OK = false;

  ifstream OpenFile(nomeArquivoArchC.c_str());
```

```
if (!OpenFile)
  cout << "Nao foi possivel abrir o arquivo " + nomeArquivoArchC << endl;
else
{
  ifstream teste(nomeArquivoGALS.c_str());
  if (teste)
  {
    cout << "Ja existe um arquivo chamado " + nomeArquivoGALS + ". Substituir?" << endl;
    cout << "(digite S para \"sim\", ou outra coisa para \"nao\"): ";
    char resp;
    cin >> resp;
    if (resp == 'S' || resp == 's')
    {
      OK = true;
    }
    else
      OK = false;
  }
  else
    OK = true;
}

if(OK)
{
  lexico.setInput(OpenFile);
  try
  {
    sintatico.parse(&lexico, &semantico);
    OpenFile.close();
    instrucoes = semantico.getVetorInstrucoes();
    formatos = semantico.getVetorFormatos();

    gramatica = new Gramatica();
    gramatica->montaGramatica(formatos, instrucoes);
    gramatica->geraArquivoGALS(nomeArquivoGALS);

    montaListaInstr();
  }
  catch ( LexicalError &e )
  {
    cout << "Erro lexico!\n"  << e.getMessage() << ", posicao: " << e.getPosition();
  }
  catch ( SyntaticError &e )
  {
    cout << "Erro sintatico!\n" << e.getMessage() << ", posicao: " << e.getPosition();
  }
  catch ( SemanticError &e )
  {
    cout << "Erro semantico!\n" << e.getMessage() << ", posicao: " << e.getPosition();
```

```
    }
  }
} //fim do método
//---------------------------------------------------------------------------
void Main::montaListaInstr()
{
  ofstream arquivo("montador/ListaInstrucoes.cpp");

  arquivo << "#include \"ListaInstrucoes.h\"\n\n";
  //construtor, que inicializa a lista de instrucoes
  arquivo << "ListaInstrucoes::ListaInstrucoes()\n"
          << "{\n";
  arquivo << "  vector<CampoInstr> campos;\n"
          << "  Instrucao* instr;\n"
          << "  vector<string> ordemCampos;\n";
  for (int i = 0; i < instrucoes.size(); i++)
  {
    for (int j = 0; j < instrucoes[i]->getFormato()->campos.size(); j++)
    {
      // Criando o vetor de campos da instrução
      arquivo << "  campos.push_back(*new CampoInstr("
              << "\"" << instrucoes[i]->getFormato()->campos[j].getNome() << "\"" << ","
              << instrucoes[i]->getFormato()->campos[j].getPosInicio() << ","
              << instrucoes[i]->getFormato()->campos[j].getTamanho() << ","
              << "\"" << instrucoes[i]->getFormato()->campos[j].getValor() << "\"" << ","
              << instrucoes[i]->getFormato()->campos[j].getComSinal() << ","
              << instrucoes[i]->getFormato()->campos[j].getCampoUsuario() << ","
              << instrucoes[i]->getFormato()->campos[j].getCampoEndereco() << "));\n";
    }
    //criando a instrução e adicionando no vetor
    arquivo << "  instr = new Instrucao("
            << "\"" << instrucoes[i]->getMnemonico() << "\"" << ","
            << "new FormatoInstr("
            << "\"" << instrucoes[i]->getFormato()->getNome() << "\"" << ","
            << "campos));\n";
    arquivo << "  instr->setAssembly("
            << "\"" << instrucoes[i]->getAssembly() << "\");\n";
    arquivo << "  instr->setTipoBranch((Instrucao::TipoBranch)"
            << instrucoes[i]->getTipoBranch() << ");\n";

    for (int j = 0; j < instrucoes[i]->getOrdemCamposAsm().size(); j++)
    {
      arquivo << "  ordemCampos.push_back("
              << "\"" << instrucoes[i]->getOrdemCamposAsm()[j] << "\");\n";
    }
    arquivo << "  instr->setOrdemCamposAsm("
            << "ordemCampos);\n";

    arquivo << "  instr->setMnemonico(\""
```

```
            << instrucoes[i]->getMnemonico() << "\");\n";
    arquivo << "  instrucoes.push_back(*instr);\n";
    arquivo << "  ordemCampos.clear();\n";
    arquivo << "  campos.clear();\n\n";

  }
  arquivo << "}\n\n";

  //método getListaInstr()
  arquivo << "vector<Instrucao>& ListaInstrucoes::getListaInstr()\n"
          << "{\n"
          << "  return instrucoes;\n"
          << "}\n";

  arquivo.close();
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
  if (argc < 3)
    cout << "Uso: gerador [arquivo ArchC] [arquivo GALS] " << endl;
  else
    Main* m = new Main(string(argv[1]), string(argv[2]));

  return 0;
}
//---------------------------------------------------------------------------
```

5.2.2 Arquivo main.h

```
#ifndef MAIN_H
#define MAIN_H

#include "Lexico.h"
#include "Sintatico.h"
#include "Semantico.h"
#include "Instrucao.h"
#include "FormatoInstr.h"
#include "Gramatica.h"

class Main
{
  public:
    Main(string nomeArquivoArchC, string nomeArquivoGALS);

  private:

    //Gerados Pelo GALS
    Lexico lexico;
    Sintatico sintatico;
```

```
    Semantico semantico;

    vector<Instrucao*> instrucoes;
    vector<FormatoInstr*> formatos;
    Gramatica* gramatica;

    void montaListaInstr();
};

#endif
```

5.2.3 Arquivo semantico.cpp

```
#include "Semantico.h"
#include "Constants.h"
#include <iostream>
#include <string>

using namespace std;

//------------------------------------------------------------------------------
Semantico::Semantico()
{
  contPosCampos = 0;
}
//------------------------------------------------------------------------------
void Semantico::executeAction(int action, const Token *token) throw (SemanticError)
{
    string tokenLido = token->getLexeme();

    bool lendoNomeCampo = false;
    bool lendoTamanhoCampo = false;
    bool campoComSinal = false;

    string nomeCampo;
    string tamanhoCampo;
    vector<string> listaCampos;
    vector<string> nomeCamposTemp;

    switch (action)
    {
      //#1 Guarda ID dizendo que este é nome reservado para o identificador da descricao
      case 1:
        idDescricao = tokenLido;
        break;

      //#2 Verifica se o ID é o mesmo lido pela ação 1 (identificador da descricao)
      case 2:
        if (idDescricao != tokenLido)
          throw SemanticError("ID do construtor deve ser igual ao
```

```
      da descricao", token->getPosition());
  break;

//#3 Verifica se o formato lido existe; Se existe: Erro!
// Se não existe: insere na lista e guarda formato_atual
case 3:
  contPosCampos=0;
  if (pertenceAoVetor(nomesFormatos, tokenLido))
    throw SemanticError(string("Declaracao multipla do formato: ") +
      tokenLido, token->getPosition());
  formato_antigo = formato_atual;
  formato_atual = tokenLido;
  if(formato_antigo != formato_atual)
    nomeCamposTemp.clear();
  nomesFormatos.push_back(formato_atual);
  break;

//#4 Verifica se o campo existe; Se existe: erro!
// Se não existe: insere campo e guarda campo_atual
case 4:
{
  if (pertenceAoVetor(nomeCamposTemp, tokenLido))
    throw SemanticError(string("Declaracao multipla do campo: ") +
      tokenLido + string(" no formato ") + formato_atual, token->getPosition());
  campo_atual = tokenLido;
  nomesCampos.push_back(campo_atual);
  nomeCamposTemp.push_back(campo_atual);
  break;

}
//#5 cria novo objeto campoInstr com o nome campo_atual e tamanho lido
case 5:
{
  //atoi(tokenLido.c_str()) é o tamanho do campo
  CampoInstr* campo = new CampoInstr(campo_atual,contPosCampos,
    atoi(tokenLido.c_str()), "", false, false, false);

  //seta posicao inicial do campo
  campo->setPosInicio(contPosCampos);
  contPosCampos +=atoi(tokenLido.c_str());

  campo->setPosFim(contPosCampos-1);
  campos.push_back(*campo);
  break;

//#6 verifica char; seta sinal para o campo atual se o char for "s"
}
case 6:
  if (tokenLido == "s")
```

```
    {
      for (int i = 0; i < campos.size(); i++)
        if (campos[i].getNome() == campo_atual)
        {
          campos[i].setComSinal(true);
          break;
        }
    }
    else
    {
      throw SemanticError("Para indicar um campo sinalizado,
        use somente 's' minusculo", token->getPosition());
    }
    break;

  //#7 cria um novo formato com o nome formato_atual e com os campos lidos
  case 7:
  {
    formatos.push_back(new FormatoInstr(formato_atual,campos));
    campos.clear();
    break;
  }

  //#8 verifica se o existe o formato; se sim, armazena o nome em formato_atual
  case 8:
    if (pertenceAoVetor(nomesFormatos, tokenLido))
      formato_atual = tokenLido;
    else
      throw SemanticError(string("Formato ") + tokenLido +
        string(" especificado em ac_instr"), token->getPosition());
    break;

  //#9 verifica se a instrucao existe; se sim, erro;
  //se não, cria nova instrucao com o nome lido e formato_atual
  case 9:
    if (pertenceAoVetor(nomesInstrucoes, tokenLido))
      throw SemanticError(string("Declaracao multipla da instrucao: ") +
        tokenLido + string(" do formato ") + formato_atual, token->getPosition());
    for (unsigned int i = 0; i < formatos.size(); i++)
      if (formatos[i]->getNome() == formato_atual)
      {
        instrucoes.push_back(new Instrucao(tokenLido,
          new FormatoInstr(*formatos[i])));
        nomesInstrucoes.push_back(tokenLido);
        break;
      }
    break;

//#10 Verificar se ID é uma instrucao e ainda não foi setado o seu assemlby.
```

```
// Se é instrucao e não foi setado o seu assembly:
// guarda "instr_atual", marcar instrucao como tendo assemlby setado
// Se não é instrucao ou já foi setado o assembly: erro
case 10:
  if (!pertenceAoVetor(nomesInstrucoes, tokenLido))
    throw SemanticError(string("Instrucao indefinida: ") +
      tokenLido, token->getPosition());

    if (pertenceAoVetor(instrucoesComAssembly, tokenLido))
    throw SemanticError(string("Assembly da instrucao ") + tokenLido +
      string(" sendo setado mais de uma vez"), token->getPosition());
    instr_atual = tokenLido;
    instrucoesComAssembly.push_back(instr_atual);
  break;

  //#11 setar o assembly da instr_atual para o ASSEMBLY lido
  case 11:
  {
    string assemblyInstr;
    for (unsigned int i = 2; i < tokenLido.length()-2; i++)
      assemblyInstr += tokenLido.at(i);

    for (unsigned int i = 0; i < instrucoes.size(); i++)
      if (instrucoes[i]->getNome() == instr_atual)
      {
        instrucoes[i]->setAssembly(assemblyInstr);
        //instrucoes[i]->setCamposUsuario();
      }
  }
  break;

//#12 Verificar se ID é uma instrucao e ainda não foi setado o seu decoder.
// Se é instrucao e não foi setado o seu decoder:
// guarda "instr_atual", marcar instrucao como tendo decoder setado
// Se não é instrucao ou já foi setado o decoder: erro
case 12:
  if (!pertenceAoVetor(nomesInstrucoes, tokenLido))
    throw SemanticError(string("Instrucao indefinida: ") +
      tokenLido, token->getPosition());

    if (pertenceAoVetor(instrucoesComDecoder, tokenLido))
    throw SemanticError(string("Decoder da instrucao ") + tokenLido +
      string(" sendo setado mais de uma vez"), token->getPosition());
    instr_atual = tokenLido;
    instrucoesComDecoder.push_back(instr_atual);
  break;

 //#13 verifica se o campo lido faz parte do formato da instr_atual
 // e se seu valor não foi setado
```

```
 //se o campo lido faz parte do formato da instr_atual e o valor
 // não foi setado, guarda campo_atual
 //se o campo não faz parte ou valor já setado, erro.
case 13:
{
  bool existeCampo = false;
    for (int i = 0; i < instrucoes.size(); i++)
    {
      if (instrucoes[i]->getNome() == instr_atual)
      {
        int j=0;
        while(!existeCampo && j<instrucoes[i]->getFormato()->campos.size())
        {
          if (instrucoes[i]->getFormato()->campos[j].getNome() == tokenLido)
            existeCampo = true;
          j++;
        }
      }
    }
    if (!existeCampo){
      throw SemanticError("Campo indefinido: " + tokenLido, token->getPosition());
    }
    if(pertenceAoVetor(camposValorSetado, tokenLido))
      throw SemanticError(string("Valor do campo ") + tokenLido +
         string(" sendo setado mais de uma vez"), token->getPosition());
    campo_atual = tokenLido;
  }
  break;

  case 14:
  {
    for (int i = 0; i < instrucoes.size(); i++)
    {
      if (instrucoes[i]->getNome() == instr_atual)
      {
        for (int j = 0; j < instrucoes[i]->getFormato()->campos.size(); j++)
          if (instrucoes[i]->getFormato()->campos[j].getNome() == campo_atual)
          {
            instrucoes[i]->getFormato()->campos[j].setValor(tokenLido);

            break;
          }
      }
    }
    break;
  }

  //#15 Seta uma flag dizendo se é relativo ou absoluto
  case 15:
```

```
        if(tokenLido == ("ac_addr_mode_R"))
          tipoBranch = Instrucao::RELATIVO;
        else
          tipoBranch = Instrucao::ABSOLUTO;
        break;

      //#16 Armazena o nome da instrução e seta como sendo de branch
      case 16:{
        instr_atual = tokenLido;
        Instrucao& inst = ProcuraInstrucao(instr_atual);
        inst.setTipoBranch(tipoBranch);
        break;
      }

      //#17 Seta o campo lido como sendo de endereço
      case 17:
        Instrucao& inst = ProcuraInstrucao(instr_atual);
        CampoInstr& campo = ProcuraCampo(inst,tokenLido);
        campo.setCampoEndereco(true);
      break;
    }
}
//---------------------------------------------------------------------------
Instrucao& Semantico::ProcuraInstrucao(string& nomeInstrucao)
{
  for (int i = 0; i < instrucoes.size(); i++)
    if (instrucoes[i]->getNome() == nomeInstrucao)
      return *instrucoes[i];
}
//---------------------------------------------------------------------------
CampoInstr& Semantico::ProcuraCampo(Instrucao& instrucao, string& nomeCampo)
{
  for (int i = 0; i < instrucao.getFormato()->campos.size(); i++)
    if (instrucao.getFormato()->campos[i].getNome() == nomeCampo)
      return instrucao.getFormato()->campos[i];
}
//---------------------------------------------------------------------------
vector<string> Semantico::retornaCampos(string listaCampos)
{
  vector<string> retorno;
  string atual;

  for (unsigned int i = 1; i < listaCampos.length()-1; i++)
  {
    atual += listaCampos.at(i);
    if (listaCampos.at(i+1) == '%' || listaCampos.at(i+1) == '\"')
    {
      retorno.push_back(atual);
      atual = "";
```

```
    }
  }
  return retorno;
}
//---------------------------------------------------------------------------
vector<Instrucao*> Semantico::getVetorInstrucoes()
{
  return instrucoes;
}
//---------------------------------------------------------------------------
vector<FormatoInstr*> Semantico::getVetorFormatos()
{
  return formatos;
}
//---------------------------------------------------------------------------
void Semantico::imprimeInstrucoes()
{
  cout << "\n-----Instrucoes-----\n";
  for (unsigned int i = 0; i < instrucoes.size(); i++)
  {
    cout << "**************************\n";
    cout << "- Mnemonico: " + instrucoes[i]->getNome() + "\n";
    cout << "- Formato: " + instrucoes[i]->getFormato()->getNome() + "\n";
    cout << "- Assembly: " + instrucoes[i]->getAssembly() + "\n";
    cout << "\n";
    cout << "- Campos:\n";
    instrucoes[i]->getFormato()->imprimeCampos();
  }
}
//---------------------------------------------------------------------------
bool Semantico::pertenceAoVetor (vector<string>& vetor, string elemento)
{
  for (int i = 0; i < vetor.size(); i++)
    if (vetor[i] == elemento)
      return true;
  return false;
}
```

5.2.4 Arquivo semantico.h

```
#ifndef SEMANTICO_H
#define SEMANTICO_H

#include "Token.h"
#include "SemanticError.h"
#include "CampoInstr.h"
#include "FormatoInstr.h"
#include "Instrucao.h"
#include <vector>
```

```
class Semantico
{
  public:
    Semantico();
    //Método gerado pelo GALS
    void executeAction(int action, const Token *token) throw (SemanticError );


    void imprimeInstrucoes();
    vector<Instrucao*> getVetorInstrucoes();
    vector<FormatoInstr*> getVetorFormatos();

  private:
    vector<CampoInstr> campos;
    vector<FormatoInstr*> formatos;
    vector<Instrucao*> instrucoes;
    string formato_atual;
    string formato_antigo;
    string instr_atual;
    string campo_atual;
    string idDescricao;
    Instrucao::TipoBranch tipoBranch;
    vector<string> nomesFormatos;
    vector<string> nomesCampos;
    vector<string> nomesInstrucoes;
    vector<string> instrucoesComAssembly;
    vector<string> instrucoesComDecoder;
    vector<string> camposValorSetado;
    int contPosCampos; //para posicao inicial dos campos dos formatos

    //Método que quebra o string com todos os campos e seus tamanhos
    vector<string> retornaCampos (string listaCampos);
    bool pertenceAoVetor (vector<string>& vetor, string elemento);
    Instrucao& ProcuraInstrucao(string& nomeInstrucao);
    CampoInstr& ProcuraCampo(Instrucao& instrucao, string& nomeCampo);

};

#endif
```

5.2.5 Arquivo instrucao.cpp

```
#include "Instrucao.h"
#include "FuncoesGerais.h"


Instrucao::Instrucao(string nome, FormatoInstr* formato)
{
  this->nome = nome;
```

```
  this->formato = formato;
  tipoBranch = NENHUM;
}
//---------------------------------------------------------------------------
void Instrucao::setNome(string nome)
{
  this->nome = nome;
}
//---------------------------------------------------------------------------
void Instrucao::setMnemonico(string mnemonico)
{
  this->mnemonico = mnemonico;
}
//---------------------------------------------------------------------------
void Instrucao::setAssembly(string assembly)
{
  this->assembly = assembly;
}
//---------------------------------------------------------------------------
void Instrucao::setFormato(FormatoInstr* formato)
{
  this->formato = formato;
}
//---------------------------------------------------------------------------
string Instrucao::getNome()
{
  return nome;
}
//---------------------------------------------------------------------------
string Instrucao::getMnemonico()
{
  return mnemonico;
}
//---------------------------------------------------------------------------
string Instrucao::getAssembly()
{
  return assembly;
}
//---------------------------------------------------------------------------
FormatoInstr* Instrucao::getFormato()
{
  return formato;
}
//---------------------------------------------------------------------------
void Instrucao::setCamposUsuario()
{
  for (int i = 0; i < ordemCamposAsm.size(); i++)
  {
    for (int j = 0; j < formato->campos.size(); j++)
```

```
    {
      if (formato->campos[j].getNome() == ordemCamposAsm[i])
      {
        formato->campos[j].setCampoUsuario(true);
        break;
      }
    }
  }
}
//---------------------------------------------------------------------------
void Instrucao::addCampoOrdem(string campo)
{
  ordemCamposAsm.push_back(campo);
}
//---------------------------------------------------------------------------
void Instrucao::setOrdemCamposAsm(vector<string> ordemCampos)
{
  ordemCamposAsm = ordemCampos;
}
//---------------------------------------------------------------------------
vector<string> Instrucao::getOrdemCamposAsm()
{
  return ordemCamposAsm;
}
//---------------------------------------------------------------------------
string Instrucao::removeEspacos(string texto)
{
  string retorno;
  for (unsigned int i = 0; i < texto.length(); i++)
    if (texto.at(i) != ' ')
      retorno += texto.at(i);

  return retorno;
}
//---------------------------------------------------------------------------
void Instrucao::setTipoBranch(TipoBranch tipo)
{
  tipoBranch = tipo;
}
//---------------------------------------------------------------------------
Instrucao::TipoBranch Instrucao::getTipoBranch()
{
  return tipoBranch;
}
//---------------------------------------------------------------------------

int Instrucao::getTamanhoInstrucao() {

  int tamanhoPalavraInstr = 0;
```

```
  //contando o tamanho em bits da instrução sendo decodificada
  for (int j = 0; j < formato->campos.size(); j++)
      {
        tamanhoPalavraInstr += formato->campos[j].getTamanho();
      }
  return tamanhoPalavraInstr;

}
//---------------------------------------------------------------------------
int Instrucao::getPosicaoCampoLabel()
{
  // Procura qual campo é o campo de endereço
  int posicaoCampo = -1;
  for (int j = 0; j < formato->campos.size(); j++)
  {
    if(formato->campos[j].getCampoEndereco())
      posicaoCampo = j;
  }
  vector<string> tokensSeparados = FuncoesGerais::getTokensSeparados(assembly);
  for(int i=0; i<tokensSeparados.size(); i++)
    if(tokensSeparados[i].substr(1,tokensSeparados[i].length()-1) ==
      formato->campos[posicaoCampo].getNome())
      return i;
  return -1;

}
//---------------------------------------------------------------------------
```

5.2.6 Arquivo instrucao.h

```
#ifndef INSTRUCAO_H
#define INSTRUCAO_H

#include "FormatoInstr.h"
#include <string>
#include <iostream>

using namespace std;

/*
Classe que define as Instruções, de acordo com os formatos já definidos na Classe FormatoInstr.h
Essa classe é do Montador
*/

class Instrucao
{
  public:
    enum TipoBranch {
      RELATIVO=0,
      ABSOLUTO=1,
```

```
      NENHUM=2
    };

    //Construtor.  Todo mnemonico tem seu formato
    Instrucao(string nome, FormatoInstr* formato);

    //Sets para Mnemonico, Assembly e Formato
    void setNome(string nome);
    void setMnemonico(string mnemonico);
    void setAssembly(string assembly);
    void setFormato(FormatoInstr* formato);
    void setTipoBranch(TipoBranch tipo);
    int getTamanhoInstrucao();



    //Gets para Mnemonico, Assembly e Formato
    string getNome();
    string getMnemonico();
    string getAssembly();
    FormatoInstr* getFormato();
    TipoBranch getTipoBranch();

    vector<string> getOrdemCamposAsm();

    void setCamposUsuario();
    void addCampoOrdem(string campo);
    void setOrdemCamposAsm(vector<string> ordemCampos);
    int getPosicaoCampoLabel();

   private:
    FormatoInstr* formato;
    string nome;
    string mnemonico;
    string assembly;
    vector<string> ordemCamposAsm; // ordem que os campos aparecem no assembly da instrução

    string removeEspacos(string texto);
    TipoBranch tipoBranch;

};

#endif
```

5.2.7 Arquivo funcoesGerais.cpp

```
#include "FuncoesGerais.h"

#include <string>
```

```
//---------------------------------------------------------------------------
FuncoesGerais::FuncoesGerais()
{

}
//---------------------------------------------------------------------------
vector<string> FuncoesGerais::SeparaToken(string& token)
{
  vector<string> retorno;
  string valor;
  string caractereEsp;


  for (int j = 0; j < token.length(); j++) //percorre o token atual
  {
    if (EhCaractereEspecial(token.at(j)))
    {
      caractereEsp += token.at(j);
      retorno.push_back(caractereEsp);
      caractereEsp.clear();
    }
    else
    {
      valor += token.at(j);
      if (j != token.length()-1)
      {
        if (EhCaractereEspecial(token.at(j+1)))
        {
          retorno.push_back(valor);
          valor.clear();
        }
      }
      else
      {
        retorno.push_back(valor);
      }
    }
  }
  return retorno;
}
//---------------------------------------------------------------------------
/*ASCII
Deixa fora
48 a 57 : números de 0 a 9
65 a 90: maiúsculas
97 a 122: minúsculas
*/

bool FuncoesGerais::EhCaractereEspecial(char caractere)
```

```
{
  for (int i = 32; i < 37; i++)  //16 char
  {
    char esp = {i};
    if(esp == caractere)
      return true;
  }

  for (int i = 38; i < 48; i++)  //16 char
  {
    char esp = {i};
    if(esp == caractere)
      return true;
  }

  for (int i = 58; i < 65; i++)// 7 char
  {
    char esp = {i};
    if(esp == caractere)
      return true;
  }

  for (int i = 91; i < 97; i++) //5 char
  {
    char esp = {i};
    if(esp == caractere)
      return true;
  }

  for (int i = 123; i < 127; i++) // 4 char
  {
    char esp = {i};
    if(esp == caractere)
      return true;
  }
  //Total: 33 caracteres especiais
  return false;
}
//--------------------------------------------------------------------------------
string FuncoesGerais::ConcatenaVetor(vector<string>& tokensLinha)
{
  string retorno(tokensLinha[0]);
  int contador = 1;
  bool caractereEspecial = true;
  while( caractereEspecial && contador < tokensLinha.size()){
    if(tokensLinha[contador][0] != '-') // numero negativo
      caractereEspecial = EhCaractereEspecial(tokensLinha[contador][0]);
    else
      caractereEspecial = false;
```

```
    if(caractereEspecial) {
      retorno += tokensLinha[contador][0];
      contador++;
    }
  }
  retorno += " ";
  for(int k=contador; k<tokensLinha.size();k++)
    retorno += tokensLinha[k];
  return retorno;
}
//---------------------------------------------------------------------------

char* FuncoesGerais::StringToChar(string& str)
{
  int tamanho = str.length();
  char* retorno = new char[tamanho+1];
  for(int i=0; i<tamanho; i++)
    retorno[i] = str[i];
  retorno[tamanho] = 0;

  return retorno;
}
//---------------------------------------------------------------------------
vector<string> FuncoesGerais::getTokensSeparados(string& line)
{
  vector<string> retorno;
  string valor;
  string caractereEsp;

  char linha[1024];
  char* token;
  token = strtok(StringToChar(line), "\t ");

  while(token) {
    string stringToken(token);
    vector<string> tokensSeparados = SeparaToken(stringToken);
    for(int i=0; i<tokensSeparados.size(); i++)
      retorno.push_back(tokensSeparados[i]);
    token = strtok(NULL, "\t ");
  }

  return retorno;
}
//---------------------------------------------------------------------------
```

5.2.8 Arquivo funcoesGerais.h

```
//---------------------------------------------------------------------------

#ifndef FuncoesGeraisH
```

```
#define FuncoesGeraisH

#include <vector>

using namespace std;
class FuncoesGerais
{
  public:
    FuncoesGerais();
    static vector<string> SeparaToken(string& token);
    static string ConcatenaVetor(vector<string>& tokensLinha);
    static bool EhCaractereEspecial(char caractere);
    static char* StringToChar(string& str);
    static vector<string> getTokensSeparados(string& linha);

};

//---------------------------------------------------------------------------
#endif
```

5.2.9 Arquivo formatoInstr.cpp

```
#include "FormatoInstr.h"

//Construtor para o formato
  Instr::  Instr(string nome, vector<CampoInstr> campos)
{
  this->nome = nome;
  this->campos = campos;
}



//---------------------------------------------------------------------------
//Construtor de Cópia para ao
  Instr::  Instr(const   Instr& outro)
{
  nome = outro.nome;
  campos = outro.campos;
}
//---------------------------------------------------------------------------
  Instr   Instr::operator=(const   Instr& outro)
{
  nome = outro.nome;
  campos = outro.campos;
  return (*this);
}
//---------------------------------------------------------------------------
  Instr::~  Instr()
{
}
```

```
//---------------------------------------------------------------------------
//Set para o nome do formato
void   Instr::setNome(string nome)
{
  this->nome = nome;
}

//---------------------------------------------------------------------------
//Adiciona novo campo ao formato
void   Instr::addCampo(CampoInstr& campo)
{
  campos.push_back(campo);
}

//---------------------------------------------------------------------------
string   Instr::getNome()
{
  return nome;
}
//---------------------------------------------------------------------------


//Procura por campo específico e seta seu valor
void   Instr::setValorCampo(string campo, string valor)
{
  for (int i = 0; i < campos.size(); i++)
    if (campos[i].getNome() == campo)
     {
      campos[i].setValor(valor);
     }
}
//---------------------------------------------------------------------------

//Exibe campos Normais e Campos do Usuário
void   Instr::imprimeCampos()
{
  for (int j = 0; j < campos.size(); j++)
  {
    if (!campos[j].getCampoUsuario())
      cout << "   > " + campos[j].getNome() + " = " + campos[j].getValor() + "\n";
    else
      cout << "   > " + campos[j].getNome() + " = campo de usuario\n";
  }
}
```

5.2.10 Arquivo formatoInstr.h

```
#ifndef FORMATOINSTR_H
#define FORMATOINSTR_H
```

```
#include "CampoInstr.h"
#include <vector>
#include <string>
#include <iostream>

using namespace std;


/*
Classe que define os formatos, adicionando, para cada formato, seu nome e campos correspondentes
*/

class   Instr
{
  public:

    //Construtor de um formato, sendo uma estrutura com Nome e campos
    // (um ou mais) Instr(string nome, vector<CampoInstr> campos);


    //Construtor de Cópia - Usado para que o valor do objeto
    //  original não seja alterado. (Clone de Objeto)
      Instr(const   Instr& outro);
      Instr operator=(const   Instr& outro);
      ~  Instr();



    void setNome(string nome); //Atribui Nome ao formato
    void addCampo(CampoInstr& campo); //Insere novo campo para o formato
    string getNome(); //Gets para o Nome do
    void setValorCampo(string campo, string valor);
    //Atribui valores lidos da descrição ArchC para campos NÂO do usuário

    void imprimeCampos(); //Mostra Campos

    vector<CampoInstr> campos;

  private:
    string nome;
};

#endif
```

5.2.11 Arquivo campoInstr.cpp

```
#include "CampoInstr.h"

//Construtor de cada campo do formato
```

```
CampoInstr::CampoInstr(
  string nome,
  int posInicio,
  int tamanho,
  string valor,
  bool comSinal,
  bool campoUsuario,
  bool campoEndereco
)
{
  this->nome = nome;
  this->posInicio = posInicio;

  this->tamanho = tamanho;
  this->valor = valor;
  this->comSinal = comSinal;
  this->campoUsuario = campoUsuario;
  this->campoEndereco = campoEndereco;

}

//Sets para Nome, Tamanho e Valor de cada campo do formato
//---------------------------------------------------------------------------
void CampoInstr::setNome(string nome)
{
  this->nome = nome;
}

//---------------------------------------------------------------------------
void CampoInstr::setTamanho(int tamanho)
{
  this->tamanho = tamanho;
}
//---------------------------------------------------------------------------
void CampoInstr::setPosInicio(int pos)
{
 posInicio=pos;
}
//---------------------------------------------------------------------------
void CampoInstr::setPosFim(int pos)
{
 posFim=pos;
}
//---------------------------------------------------------------------------
int CampoInstr::getPosInicio()
{
 return posInicio;
}
//---------------------------------------------------------------------------
```

```
int CampoInstr::getPosFim()
{
 return posFim;
}
//---------------------------------------------------------------------------
void CampoInstr::setValor(string valor)
{
  this->valor = valor;
}
//---------------------------------------------------------------------------
void CampoInstr::setComSinal(bool comSinal)
{
  this->comSinal = comSinal;
}



//Gets para Nome, Tamanho e Valor de cada campo do formato
//---------------------------------------------------------------------------
string CampoInstr::getNome()
{
  return nome;
}
//---------------------------------------------------------------------------
int CampoInstr::getTamanho()
{
  return tamanho;
}
//---------------------------------------------------------------------------
string CampoInstr::getValor()
{
  return valor;
}


/*Set e Get para Campos do Usuário, nos quais seus valores serão atribuídos a
partir do código assembly, e não da descrição ArchC*/
//---------------------------------------------------------------------------
void CampoInstr::setCampoUsuario(bool campoUsr)
{
  campoUsuario = campoUsr;
}
//---------------------------------------------------------------------------
bool CampoInstr::getCampoUsuario()
{
  return campoUsuario;
}
//---------------------------------------------------------------------------
```

```
bool CampoInstr::getComSinal()
{
  return comSinal;
}
//---------------------------------------------------------------------------
void CampoInstr::setCampoEndereco(bool valor)
{
  campoEndereco = valor;
}

//---------------------------------------------------------------------------
bool CampoInstr::getCampoEndereco()
{
  return campoEndereco;
}

//---------------------------------------------------------------------------
```

5.2.12 Arquivo campoInstr.h

```
#ifndef CAMPOINSTR_H
#define CAMPOINSTR_H

#include <string>

using namespace std;

/*
Classe que define os campos de cada formato
*/
class CampoInstr
{
  public:

    //Construtor para os campos do formato
    CampoInstr
    (
      string nome,
      int posInicio,
      int tamanho,
      string valor,
      bool comSinal,
      bool campoUsuario,
      bool campoEndereco
    );

    //Sets para Nome, Tamanho e Valor para cada campo do formato
    void setNome(string nome);
    void setTamanho(int tamanho);
    void setValor(string valor);
```

```
    void setComSinal(bool comSinal);
    void setCampoEndereco(bool valor);

    //seta pos inicio e fim para cada campo de cada formato
    void setPosInicio(int pos);
    void setPosFim(int pos);

    //Gets para Nome, Tamanho e Valor para cada campo do formato
    string getNome();
    int getTamanho();

    //pega pos inicio e fim para cada campo de cada formato
    int getPosInicio();
    int getPosFim();

    bool getCampoEndereco();

    string getValor();


    /*Define se o campo é do usuário ou não. Campos como valores e
   Regra: Campos que possuem valores lidos da descrição ArchC NÂO são
   considerados campos de usuários.
   Campos restantes, são campos do usuário, ou seja, seus valores vem
   da aplicação(código assembly)
   */
    void setCampoUsuario(bool campoUsr);
    bool getCampoUsuario();
    bool getComSinal();

  private:
    //Atributos para os campos do formato
    string nome;
    int tamanho;
    int posInicio;
    //posicao de inicio do campo dentro do formato, da direita para esquerda  Ex: 15 .. 0
    int posFim;    //posicao de fim do campo dentro do formato , da direita para esquerda
    string valor;
    bool comSinal;
    bool campoUsuario;
    bool campoEndereco;
};
#endif
```

5.3 Anexo II - Fontes dos arquivos comuns à todos os montadores

5.3.1 Arquivo main.cpp

```
#include "Main.h"
#include <fstream>
#include <iostream>

#include "Conversor.h"

Main::Main(string nomeArquivoAsm, string nomeArquivoHex, int tamanhoPalavraDados)
{
  ifstream OpenFile(nomeArquivoAsm.c_str());
  lexico.setInput(OpenFile);

  try
  {
    sintatico.parse(&lexico, &semantico);
  }
  catch ( LexicalError &e )
  {
    cout << "Erro lexico!\n"  << e.getMessage() << ", posicao: " << e.getPosition();
  }
  catch ( SyntaticError &e )
  {
    cout << "Erro sintatico!\n" << e.getMessage() << ", posicao: " << e.getPosition();
  }
  catch ( SemanticError &e )
  {
    cout << "Erro semantico!\n" << e.getMessage() << ", posicao: " << e.getPosition();
  }
  OpenFile.close();

  vector<vector<char>*> instrucoesProntas = semantico.getInstrProntas();

  long enderecoAtual = 0;

  ofstream arquivoHex(nomeArquivoHex.c_str());

  arquivoHex << ".text" << endl;
  string numBinario;
  string instrHexa;
  //escrever todas as instruções no arquivo (em formato hexadecimal)
  for (int i = 0; i < instrucoesProntas.size(); i++)
  {
    arquivoHex << Conversor::longToHex(enderecoAtual) << " ";
    for (int j = 0; j < instrucoesProntas[i]->size(); j += 4)
```

```
    {
      numBinario = instrucoesProntas[i]->at(j);
      numBinario += instrucoesProntas[i]->at(j+1);
      numBinario += instrucoesProntas[i]->at(j+2);
      numBinario += instrucoesProntas[i]->at(j+3);
      instrHexa += Conversor::bin4BitsToHex(numBinario);
    }
    int numBlocos = instrucoesProntas[i]->size()/tamanhoPalavraDados;
    //numero de blocos em que a palavra instrução será dividida
    for (int j = 0; j < instrHexa.length(); j++)
    {
      if (j != 0)
      {
        div_t resultado = div(j, instrHexa.length()/numBlocos);
        if (resultado.rem == 0)
        {
          arquivoHex << " ";
          arquivoHex << instrHexa.at(j);
        }
        else
        {
          arquivoHex << instrHexa.at(j);
        }
      }
      else
      {
        arquivoHex << instrHexa.at(j);
      }
    }
    instrHexa.clear();
    enderecoAtual += instrucoesProntas[i]->size()/8;
    arquivoHex << endl;
  }

  arquivoHex.close();
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
  if (argc < 4)
    cout << "Uso: montador [arquivo Assembly] [arquivo .hex]
      [tamanho (em bits) da palavra de dados]" << endl;
  else
    Main* m = new Main(string(argv[1]), string(argv[2]), atoi(argv[3]));

  return 0;
}
```

5.3.2 Arquivo main.h

```
#ifndef MAIN_H
#define MAIN_H

#include "ListaInstrucoes.h"
#include "Lexico.h"
#include "Sintatico.h"
#include "Semantico.h"

class Main
{
  public:
    Main(string nomeArquivoAsm, string nomeArquivoHex, int tamanhoPalavra);

  private:
    //Gerados Pelo GALS
    Lexico lexico;
    Sintatico sintatico;
    Semantico semantico;

};
#endif
```

5.3.3 Arquivo semantico.cpp

```
#include "Semantico.h"
#include "Constants.h"
#include "Conversor.h"
#include <iostream>
#include <string>
#include <stdlib.h>


using namespace std;

//----------------------------------------------------------------------------
Semantico::Semantico()
{
  listaInstr = new ListaInstrucoes();
  indexCampoAtual = -1;
  instrucao = NULL;
}
//----------------------------------------------------------------------------
void Semantico::executeAction(int action, const Token *token) throw (SemanticError)
{
    string tokenLido = token->getLexeme();

    string nomeCampo;
    string tamanhoCampo;
```

```
vector<string> listaCampos;
vector<string> nomeCamposTemp;

switch (action)
{
  case 1:
    break;

  case 2: //Leitura do VALUE
  {
    int tamanhoCampoAtual;
    bool campoComSinal;
    indexCampoAtual++;

    //encontrar qual é a instrução sendo decodificada
    for (int i = 0; i < listaInstr->getListaInstr().size(); i++)
    {
      if (listaInstr->getListaInstr()[i].getMnemonico() == instr_atual)
      {
        //pegar o nome do campo atual (conforme a sua ordem no assembly da instrução)
        string nomeCampoAtual =
          listaInstr->getListaInstr()[i].getOrdemCamposAsm()[indexCampoAtual];

        for (int j=0;  j <
          listaInstr->getListaInstr()[i].getFormato()->campos.size(); j++)
        {
          if (nomeCampoAtual ==
            listaInstr->getListaInstr()[i].getFormato()->campos[j].getNome())
          {
           tamanhoCampoAtual =
             listaInstr->getListaInstr()[i].getFormato()->campos[j].getTamanho();
           campoComSinal =
             listaInstr->getListaInstr()[i].getFormato()->campos[j].getComSinal();
           break;
          }
        }
         //passa tamanho, valor e sinal
        if (tokenLido.length() > 1)
        {
          if (tokenLido.at(1) == 'X' || tokenLido.at(1) == 'x')
          {
              string x = Conversor::hexToBin(tokenLido);
            if (!valorHexaValido(tamanhoCampoAtual,x))
            {
              cout << "Lexema: " << token->getLexeme() << endl;
              throw SemanticError("Valor Invalido para o campo " +
               nomeCampoAtual, token->getPosition());
            }
            else
```

```
            adicionaCampoUsuario(
              listaInstr->getListaInstr()[i], nomeCampoAtual, tokenLido);
        }
        else
        {
          if(!valorDecValido(
            tamanhoCampoAtual,Conversor::strToLong(tokenLido),campoComSinal))
          {
            cout << "Lexema: " << token->getLexeme() << endl;
            throw SemanticError("Valor Invalido para o campo " +
              nomeCampoAtual, token->getPosition());
          }
          else
            adicionaCampoUsuario(listaInstr->getListaInstr()[i],
              nomeCampoAtual, tokenLido);
        }
      }
      else
      {
        adicionaCampoUsuario(listaInstr->getListaInstr()[i],
          nomeCampoAtual, tokenLido);
      }
    }
  }
  break;
}
case 3:
  indexCampoAtual = -1;
  tamanhoPalavraInstr = 0;
  instr_atual = tokenLido;

  for (int i = 0; i < listaInstr->getListaInstr().size(); i++)
  {
    if (listaInstr->getListaInstr()[i].getMnemonico() == instr_atual)
    {
      //contando o tamanho em bits da instrução sendo decodificada
      for (int j = 0; j <
        listaInstr->getListaInstr()[i].getFormato()->campos.size(); j++)
      {
        tamanhoPalavraInstr +=
          listaInstr->getListaInstr()[i].getFormato()->campos[j].getTamanho();
      }
      instrucao = new vector<char>;
      for (int k = 0; k < tamanhoPalavraInstr; k++)
      {
        instrucao->push_back('0');
      }
      adicionaCamposNaoUsuario(listaInstr->getListaInstr()[i]);
      break;
```

```
          }
        }
        break;

      case 4:
        //adiciona a instrução decodificada no vetor de instruções prontas
        instrucoesProntas.push_back(instrucao);
    }
}
//---------------------------------------------------------------------------
void Semantico::adicionaCamposNaoUsuario(Instrucao& instr)
{
  for (int j = 0; j < instr.getFormato()->campos.size(); j++)
  {
    //cout << "Campo: " << instr.getFormato()->campos[j].getNome() << endl;
    if (!instr.getFormato()->campos[j].getCampoUsuario())
    {
      int inicio = instr.getFormato()->campos[j].getPosInicio();
      int fim = inicio + instr.getFormato()->campos[j].getTamanho()-1;
      string valorCampo = instr.getFormato()->campos[j].getValor();
      long valor = Conversor::strToLong(valorCampo);

      string valorBinario = Conversor::longToBin(
        valor, instr.getFormato()->campos[j].getTamanho());

      int bitAtual = 0;
      for (int i = inicio; i <= fim; i++)
      {
        instrucao->insert(instrucao->begin()+i, valorBinario.at(bitAtual));
        instrucao->erase(instrucao->begin()+i+1);
        bitAtual++;
      }
    }
  }
}
//---------------------------------------------------------------------------
void Semantico::adicionaCampoUsuario(Instrucao& instr, string nomeCampo, string valor)
{
  for (int j = 0; j < instr.getFormato()->campos.size(); j++)
  {
    if (instr.getFormato()->campos[j].getNome() == nomeCampo)
    {
      int inicio = instr.getFormato()->campos[j].getPosInicio();
      int fim = inicio + instr.getFormato()->campos[j].getTamanho()-1;
      long valorInteiro = Conversor::strToLong(valor);

      string valorBinario = Conversor::longToBin(
        valorInteiro, instr.getFormato()->campos[j].getTamanho());
      int bitAtual = 0;
```

```
      for (int i = inicio; i <= fim; i++)
      {
        instrucao->insert(instrucao->begin()+i, valorBinario.at(bitAtual));
        instrucao->erase(instrucao->begin()+i+1);
        bitAtual++;
      }
    }
  }
}
//---------------------------------------------------------------------------
string Semantico::completaZeros(string valor, int tamanhoCampo)
{
  int numZeros = tamanhoCampo - valor.length();
  string retorno;
  retorno = valor;

  for (int i = 1; i <= numZeros; i++)
  {
    retorno = "0" + retorno;
  }
  return retorno;
}
//---------------------------------------------------------------------------
vector<vector<char>*> Semantico::getInstrProntas()
{
  return instrucoesProntas;
}
//---------------------------------------------------------------------------

bool Semantico::valorHexaValido(int tamanhoCampo, string& valor)
//varifica se o valor de um campo é valido, de acordo o tamanho do campo em bits
{
  if (tamanhoCampo < valor.length())
    return false;
  return true;
}
//---------------------------------------------------------------------------
bool Semantico::valorDecValido(int tamanhoCampo, int valor, bool campoComSinal)
{
  int valorMaximo, valorMinimo;

  if (!campoComSinal)
  {
    valorMaximo = pow((double)2,tamanhoCampo)-1;
    valorMinimo = 0;
  }
  else
  {
    valorMaximo = pow((double)2,tamanhoCampo)-1;
```

```
    valorMinimo = -1 * pow((double)2,tamanhoCampo);
  }

  if ((valor > valorMaximo) || (valor < valorMinimo))
    return false;

  return true;
}
```

5.3.4 Arquivo semantico.h

```
#ifndef SEMANTICO_H
#define SEMANTICO_H

#include "Token.h"
#include "SemanticError.h"
#include "CampoInstr.h"
#include "FormatoInstr.h"
#include "Instrucao.h"
#include "ListaInstrucoes.h"
#include <vector>
#include <math.h>

class Semantico
{
  public:
    Semantico();
    //Método gerado pelo GALS
    void executeAction(int action, const Token *token) throw (SemanticError );
    vector<vector<char>*> getInstrProntas();

  private:
    ListaInstrucoes* listaInstr;
    string instr_atual;
    int tamanhoPalavraInstr; //tamanho da palavra da instrução atual
    vector<vector<char>*> instrucoesProntas;
    vector<char>* instrucao;
    int indexCampoAtual;

    void adicionaCamposNaoUsuario(Instrucao& instr);
    void adicionaCampoUsuario(Instrucao& instr, string nomeCampo, string valor);
    string completaZeros(string valor, int tamanhoCampo);
    //verifica se o valor de um campo é valido, de acordo o tamanho do campo em bits
    bool valorHexaValido(int tamanhoCampo, string& valor);
    bool valorDecValido(int tamanhoCampo, int valor, bool campoComSinal);
};

#endif
```

5.3.5 Arquivo conversor.cpp

```
#include "Conversor.h"

#include "stdio.h"

#include <sstream>
#include <iostream>


string Conversor::longToBin(long num, int nrBits)
{
  string retorno;

  for(int i=0; i<nrBits; i++)
  {
    unsigned valor = num&(1<<i);
    if(valor == 0)
      retorno = "0" + retorno;
    else
      retorno = "1" + retorno;
  }

  return retorno;
}
//---------------------------------------------------------------------------
string Conversor::longToHex(long num)
{
  long quociente;
  long resto;
  ldiv_t resultado;
  string retorno;


  do
  {
    resultado = ldiv(num, 16);
    quociente = resultado.quot;
    resto = resultado.rem;
    string valorHexa = decToHex(resto);
    retorno = valorHexa + retorno;
    num = quociente;
  } while (quociente != 0);


  return retorno;
}
//---------------------------------------------------------------------------
string Conversor::bin4BitsToHex(string& bin)
{
```

```
if (bin == "0000")
  return "0";

if (bin == "0001")
  return "1";

if (bin == "0010")
  return "2";

if (bin == "0011")
  return "3";

if (bin == "0100")
  return "4";

if (bin == "0101")
  return "5";

if (bin == "0110")
  return "6";

if (bin == "0111")
  return "7";

if (bin == "1000")
  return "8";

if (bin == "1001")
  return "9";

if (bin == "1010")
  return "A";

if (bin == "1011")
  return "B";

if (bin == "1100")
  return "C";

if (bin == "1101")
  return "D";

if (bin == "1110")
  return "E";

if (bin == "1111")
  return "F";

return "erro";
```

```
}
//---------------------------------------------------------------------------
string Conversor::decToHex(long dec)
{
  if (dec < 10)
  {
    stringstream retorno;
    retorno << dec;
    return retorno.str();
  }
  else
  {
    switch (dec)
    {
      case 10:
      return "A";

    case 11:
      return "B";

    case 12:
      return "C";

    case 13:
      return "D";

    case 14:
      return "E";

    case 15:
      return "F";
    }
  }

  return "erro";
}
//---------------------------------------------------------------------------
long Conversor::strToLong (string& valor, bool sinalizado, int tamanhoCampo)
{
  if (valor.length() > 1)
  {
     if ((valor.at(1) == 'x') || (valor.at(1) == 'X'))
     {
       if (sinalizado)
       {
         string binario = longToBin(strtol(valor.c_str(), NULL, 16),tamanhoCampo);

         if (binario.at(0) == '1')
         {
```

```
            long decimal = strtol(valor.c_str(), NULL, 16);
            long complemento = ~decimal;
            return complemento;

          }
          else
          {
            return strtol(valor.c_str(), NULL, 16);
          }
        }
        else
        {
          return strtol(valor.c_str(), NULL, 16);
        }
      }
      else
      {
        return strtol(valor.c_str(), NULL, 10);
      }
  }
  else
  {
    return strtol(valor.c_str(), NULL, 10);
  }
}
//---------------------------------------------------------------------------
long Conversor::strToLong (string& valor)
{
  if (valor.length() > 1)
  {
     if ((valor.at(1) == 'x') || (valor.at(1) == 'X'))
     {
       return strtol(valor.c_str(), NULL, 16);
     }
     else
     {
       return strtol(valor.c_str(), NULL, 10);
     }
  }
  else
  {
    return strtol(valor.c_str(), NULL, 10);
  }
}
//---------------------------------------------------------------------------
string Conversor::hexToBin(string& hex)
{
  int inicio = 0;
  if (hex.length() > 1)
```

```
{
  if ((hex.at(1) == 'x') || (hex.at(1) == 'X'))
  {
    inicio = 2;
  }
}

string bin;
for (int i = inicio; i < hex.length(); i++)
{
  if (hex.at(i) == '0')
  {
    bin = bin + "0000";
  }
  else if (hex.at(i) == '1')
  {
    bin = bin + "0001";
  }
  else if (hex.at(i) == '2')
  {
    bin = bin + "0010";
  }
  else if (hex.at(i) == '3')
  {
    bin = bin + "0011";
  }
  else if (hex.at(i) == '4')
  {
    bin = bin + "0100";
  }
  else if (hex.at(i) == '5')
  {
    bin = bin + "0101";
  }
  else if (hex.at(i) == '6')
  {
    bin = bin + "0110";
  }
  else if (hex.at(i) == '7')
  {
    bin = bin + "0111";
  }
  else if (hex.at(i) == '8')
  {
    bin = bin + "1000";
  }
  else if (hex.at(i) == '9')
  {
    bin = bin + "1001";
```

```
    }
    else if (hex.at(i) == 'A' || hex.at(i) == 'a')
    {
      bin = bin + "1010";
    }
    else if (hex.at(i) == 'B' || hex.at(i) == 'b')
    {
      bin = bin + "1011";
    }
    else if (hex.at(i) == 'C' || hex.at(i) == 'c')
    {
      bin = bin + "1100";
    }
    else if (hex.at(i) == 'D' || hex.at(i) == 'd')
    {
      bin = bin + "1101";
    }
    else if (hex.at(i) == 'E' || hex.at(i) == 'e')
    {
      bin = bin + "1110";
    }
    else if (hex.at(i) == 'F' || hex.at(i) == 'f')
    {
      bin = bin + "1111";
    }
  }

  int cont=0;
  for (int i = 0; i<bin.length(); i++)
  {
    if(bin.at(i) == '1')
      break;
    cont++;
  }

  return bin.substr(cont,bin.length());
}
```

5.3.6 Arquivo conversor.h

```
#ifndef CONVERSOR_H
#define CONVERSOR_H

#include <string>

using namespace std;

class Conversor
{
```

```
  public:
    static string longToBin(long num, int nrBits);
    static string longToHex(long num);
    static string bin4BitsToHex(string& bin);
    static long strToLong (string& valor, bool sinalizado, int tamanhoCampo);
    static long strToLong (string& valor);
    static string decToHex(long dec);
    static string hexToBin(string& hex);
};

#endif
```

5.3.7 Arquivo listaInstrucoes.h

```
#ifndef LISTAINSTR_H
#define LISTAINSTR_H

#include "Instrucao.h"
#include "CampoInstr.h"
#include "FormatoInstr.h"
#include <vector>
#include <string>

using namespace std;

class ListaInstrucoes
{
  public:
    ListaInstrucoes();

    vector<Instrucao>& getListaInstr();

  private:
    vector<Instrucao> instrucoes;
};

#endif
```

5.4 Anexo III - Fontes do Pré-Processador

5.4.1 Arquivo Main.cpp

```
//---------------------------------------------------------------------------

#include "Main.h"
#include <iostream>
#include <fstream>

#include "Sintatico.h"
```

```
#include "Constants.h"
#include "Lexico.h"

#ifndef GERADOR
#include "PseudoInstrucao.h"
#include "PreProcessor.h"
#include "Macro.h"
#include "PalavraReservada.h"
#endif


//------------------------------------------------------------------------
int main(int argc, char* argv[])
{
  #ifdef GERADOR
  if (argc < 2)
  {
    cout << "Uso: geradorPP [arquivo PP]" << endl;
    return 0;
  }
  #endif
  #ifndef GERADOR
  if (argc < 4)
  {
    cout << "Uso: PP [arquivo PP] [Assembly entrada] [Assembly saida]" << endl;
    return 0;
  }
  ifstream arquivoEntradaAsm(argv[2]);
  ofstream arquivoSaidaAsm(argv[3]);
  if(!arquivoEntradaAsm){
    cout << "Arquivo " << argv[2] << "nao existe" << endl;
    return 0;
  }
  #endif

  string nomeArquivo(argv[1]);

  PreProcessador::Lexico lexico;
  PreProcessador::Semantico semantico;
  PreProcessador::Sintatico sintatico;

  ifstream arquivo(nomeArquivo.c_str());
  if(!arquivo){
    cout << "Arquivo " << argv[1] << "nao existe" << endl;
    return 0;
  }

  lexico.setInput(arquivo);
  #ifdef GERADOR
```

```
    semantico.setCriarArquivoAddressGenerator(true);
  #endif

  try
  {
    sintatico.parse(&lexico, &semantico);
    arquivo.close();
  }
  catch ( PreProcessador::LexicalError &e )
  {
    cout << "Erro lexico!\n"  << e.getMessage() << ", posicao: " << e.getPosition();
  }
  catch ( PreProcessador::SyntaticError &e )
  {
    cout << "Erro sintatico!\n" << e.getMessage() << ", posicao: " << e.getPosition();
  }
  catch ( PreProcessador::SemanticError &e )
  {
    cout << "Erro semantico!\n" << e.getMessage() << ", posicao: " << e.getPosition();
  }

  // Fazer o preProcessamento
  #ifdef GERADOR
  string nomeArquivoFuncoes("AddressGenerator.h");
  semantico.getFuncoes().geraArquivoFuncoes(nomeArquivoFuncoes);
  #endif

  #ifndef GERADOR
  // Fazer o preProcessamento
  vector<PseudoInstrucao*>& pseudos = semantico.getPseudoInstrucoes();
  vector<Macro*>& macros = semantico.getMacros();
  vector<PalavraReservada*> palavrasReservadas = semantico.getPalavrasReservadas();
  string comentario = semantico.getCommentSymbol();
  vector<string>& ignoreSymbols = semantico.getIgnoreSymbols();

  PreProcessor p(arquivoEntradaAsm, arquivoSaidaAsm,
         pseudos, macros, palavrasReservadas, ignoreSymbols, comentario);
  p.PreProcessaArquivo();
  #endif


}
```

5.4.2 Arquivo InstrucaoReal.cpp

```
#include "InstrucaoReal.h"
//---------------------------------------------------------------------------
InstrucaoReal::InstrucaoReal()
{
```

```
}
//---------------------------------------------------------------------------
void InstrucaoReal::AddToken(string token)
{
  Token tok;
  tok.token = token;
  tok.valorFixo = false;
  tokens.push_back(tok);
}

//---------------------------------------------------------------------------
void InstrucaoReal::SetValorFixo(string valor)
{
  Token token;
  token.token = valor;
  token.valorFixo = true;
  // Se é um valor fixo, então colocar em cima do token anterior
  tokens[tokens.size()-1] = token;

}
//---------------------------------------------------------------------------
Token* InstrucaoReal::getToken(string& token)
{
  for(int i=0;i<tokens.size();i++){
    Token& tokenAtual = tokens[i];
    if(tokenAtual.token == token)
      return &tokenAtual;
  }
  return NULL;
}
//---------------------------------------------------------------------------
Token* InstrucaoReal::getToken(int posicao)
{
  if(posicao >= tokens.size())
    return NULL;
  else
    return &tokens[posicao];
}
//---------------------------------------------------------------------------
int InstrucaoReal::nrTokens()
{
  return tokens.size();
}
//---------------------------------------------------------------------------
```

5.4.3 Arquivo InstrucaoReal.h

```
//---------------------------------------------------------------------------

#ifndef InstrucaoRealH
```

```
#define InstrucaoRealH

#include <vector>
#include <string>

using namespace std;

//---------------------------------------------------------------------------
struct Token
{
  string token;
  bool valorFixo;
};

//---------------------------------------------------------------------------
// O certo seria essa classe se chamar apenas de instrucao, mas como já
// existe um classe chamada instrucao no gerador...
class InstrucaoReal
{

  public:
    InstrucaoReal();
    void AddToken(string token);
    Token* getToken(int posicao);
    void SetValorFixo(string tokenLido);
    Token* getToken(string& token);
    int nrTokens();

  protected:


  private:
    vector<Token> tokens;

};


#endif
```

5.4.4 Arquivo Label.cpp

```
//---------------------------------------------------------------------------
#include "Label.h"
//---------------------------------------------------------------------------
Label::Label(string _nome)
{
  nome = _nome;
}
//---------------------------------------------------------------------------
void Label::SetNome(string value)
```

```
{
  nome = value;
}
//---------------------------------------------------------------------------
string Label::GetNome()
{
  return nome;
}
//---------------------------------------------------------------------------
void Label::SetEndereco(int value)
{
  endereco = value;
}
//---------------------------------------------------------------------------
int Label::GetEndereco()
{
  return endereco;
}
//---------------------------------------------------------------------------
```

5.4.5 Arquivo Label.h

```
//---------------------------------------------------------------------------

#ifndef
#define

#include <string>
using namespace std;
//---------------------------------------------------------------------------
class Label {

  public:
    Label(string _nome);
    void SetNome(string value);
    string GetNome();

    void SetEndereco(int value);
    int GetEndereco();

  protected:

  private:
    string nome;
    int endereco;

};

//---------------------------------------------------------------------------
```

```
#endif
```

5.4.6 Arquivo Macro.cpp

```
#include "Macro.h"
#include "InstrucaoReal.h"
#include "PseudoInstrucao.h"


void Macro::AdicionaInstrucaoReal(InstrucaoReal* instrucao)
{
  instrucoesReais.push_back(instrucao);
}

InstrucaoReal* Macro::GetInstrucaoReal(int posicao)
{
  return instrucoesReais[posicao];
}

PseudoInstrucao* Macro::GetPseudoInstrucao()
{
  return pseudoInstrucao;
}

void Macro::SetPseudoInstrucao(PseudoInstrucao* instrucao)
{
  pseudoInstrucao = instrucao;
}

int Macro::NumeroInstrucoesReais()
{
  return instrucoesReais.size();
}
```

5.4.7 Arquivo Macro.h

```
#ifndef MACRO_H
#define MACRO_H

#include <vector>
using namespace std;

class InstrucaoReal;
class PseudoInstrucao;

class Macro
{
  public:
    void AdicionaInstrucaoReal(InstrucaoReal* instrucao);
```

```
    InstrucaoReal* GetInstrucaoReal(int valor);
    PseudoInstrucao* GetPseudoInstrucao();
    void SetPseudoInstrucao(PseudoInstrucao* instrucao);
    int NumeroInstrucoesReais();

  protected:

  private:
    vector<InstrucaoReal*> instrucoesReais;

    PseudoInstrucao* pseudoInstrucao;

};

#endif
```

5.4.8 Arquivo palavraReservada.cpp

```
#include "PalavraReservada.h"

string& PalavraReservada::getPalavraReservada()
{
  return palavraReservada;
}

string& PalavraReservada::getValor()
{
  return valor;
}

void PalavraReservada::setPalavraReservada(string& _palavraReservada)
{
  palavraReservada = _palavraReservada;
}

void PalavraReservada::setValor(string& _valor)
{
  valor = _valor;
}
```

5.4.9 Arquivo palavraReservada.h

```
#ifndef PALAVRARESERVADA_H
#define PALAVRARESERVADA_H

#include <string>
using namespace std;
```

```
class PalavraReservada
{
  public:
    string& getPalavraReservada();
    string& getValor();
    void setPalavraReservada(string& _palavraReservada);
    void setValor(string& _valor);
  private:
    string palavraReservada;
    string valor;


};

#endif
```

5.4.10 Arquivo preProcessor.cpp

```
#include "PreProcessor.h"

#include <iostream>
#include "Macro.h"
#include "PalavraReservada.h"
#ifndef GERADOR
#include "ListaInstrucoes.h"
#endif

#include "TratadorDeLabels.h"
#include "FuncoesGerais.h"

//---------------------------------------------------------------------------
PreProcessor::PreProcessor(
  ifstream& _arquivoEntrada,
  ofstream& _arquivoSaida,
  vector<PseudoInstrucao*>& _instrucoes,
  vector<Macro*>& _macros,
  vector<PalavraReservada*>& _palavrasReservadas,
  vector<string>& _simbolosIgnorar,
  string& _tokenComentario
) : arquivoEntrada(_arquivoEntrada),
    arquivoSaida(_arquivoSaida),
    instrucoes(_instrucoes),
    macros(_macros),
    palavrasReservadas(_palavrasReservadas),
    simbolosIgnorar(_simbolosIgnorar),
    tokenComentario(_tokenComentario)
{
}
//---------------------------------------------------------------------------
void PreProcessor::SubstituiPalavrasReservadas()
```

```
{
  ofstream arquivoTemp("__temp__.tmp");
  char linha[1024];
  arquivoEntrada.getline(linha, sizeof(linha));
  int contador = 1;
  while (!arquivoEntrada.eof())
  {
    string stringLinha(linha);
    string linhaNova = SubstituiPalavrasReservadas(stringLinha);
    arquivoTemp << linhaNova << endl;
    arquivoEntrada.getline(linha, sizeof(linha));
  }
  arquivoTemp.close();
  arquivoEntrada.close();

}
//---------------------------------------------------------------------------
string PreProcessor::SubstituiPalavrasReservadas(string& linha)
{

  char* token;
  token = strtok(FuncoesGerais::StringToChar(linha), "\t ");
  if(token==NULL)
    return "";
  vector<string> tokensLinha;
  tokensLinha.push_back(token);
  token = strtok(NULL, "\t ");
  while (token != NULL)
  {
    string stringToken(token);
    vector<string> tokensSeparados = FuncoesGerais::SeparaToken(stringToken);
    for (int i = 0; i < tokensSeparados.size(); i++)
    {
      string& valor = tokensSeparados[i];
      tokensLinha.push_back(valor);
    }
    tokensSeparados.clear();
    token = strtok(NULL, "\t ");
  }

  if(tokensLinha[0] == tokenComentario){
    return ""; // Ignorar linha
  }
  if(tokensLinha.size() > 1) {
    if(tokensLinha[1] == tokenComentario){
      return string(tokensLinha[1]);
    }
  }
  for(int i=0; i<2 && i<tokensLinha.size(); i++)
```

```
    for(int j=0; j<simbolosIgnorar.size(); j++)
      if(tokensLinha[i] == simbolosIgnorar[j])
        tokensLinha[i] = "";

  string retorno(tokensLinha[0]);
  retorno += " ";

  int inicio = 1;
  if(tokensLinha[0][retorno.length()-2] == ':'){ // Label
    retorno += tokensLinha[1];
    inicio = 2;
    retorno += " ";
  }

/*
  if(tokensLinha.size() > 1) {
    retorno += tokensLinha[1];
    retorno += " ";
  }
  */
  bool comentario = false;
  for(int i=inicio; i<tokensLinha.size();i++)
  {
    bool mudouToken = false;
    if(tokensLinha[i] == tokenComentario){
      tokensLinha[i] == ";";
      comentario = true;
    }
    for(int j=0; j<simbolosIgnorar.size(); j++){
      if(tokensLinha[i] == simbolosIgnorar[j])
        tokensLinha[i] = "";
    }

    for(int j=0; j<palavrasReservadas.size();j++){
      if(palavrasReservadas[j]->getPalavraReservada() == tokensLinha[i])
      {
        retorno += palavrasReservadas[j]->getValor();
        mudouToken = true;
        retorno += " ";
      }
    }
    if(comentario)
      break; // Ignorar resto da linha

    if(!mudouToken)
      retorno += tokensLinha[i];
  }
  return retorno;
}
```

```
//---------------------------------------------------------------------------
void PreProcessor::PreProcessaArquivo()
{
  SubstituiPalavrasReservadas();

  ifstream arquivoEntradaTemp("__temp__.tmp");
  ofstream arquivoSaidaTemp("__temp2_.tmp");

  char linha[1024];
  arquivoEntradaTemp.getline(linha, sizeof(linha));
  while (!arquivoEntradaTemp.eof())
  {
    string stringLinha(linha);
    string linhaNova = ParseLine(stringLinha);
    arquivoEntradaTemp.getline(linha, sizeof(linha));
    arquivoSaidaTemp << linhaNova << endl;
  }
  arquivoEntradaTemp.close();
  arquivoSaidaTemp.close();

#ifndef GERADOR
  ListaInstrucoes lista;

  TratadorDeLabels tratadorDeLabels(&lista, this);
  tratadorDeLabels.ResolveLabels("__temp2_.tmp", arquivoSaida);
#endif



}
//---------------------------------------------------------------------------
string PreProcessor::FazTrocaPseudo(
  PseudoInstrucao* pseudo,
  vector<string>& tokensLinha,
  bool temLabel
)
{
  return FazTroca(pseudo, pseudo->getInstrucaoReal(), tokensLinha, temLabel);
}
//---------------------------------------------------------------------------
string PreProcessor::FazTroca(
  PseudoInstrucao* pseudo,
  InstrucaoReal* real,
  vector<string>& tokensLinha,
  bool temLabel
)
{

  string retorno;
  vector<string> saida;
```

```
  InstrucaoReal* instrucao = real;
  unsigned int tamanho = instrucao->nrTokens();
  saida.resize(tamanho);

  // Copiar vetor
  for(int i=0;i<tamanho;i++){
    string& token = instrucao->getToken(i)->token;
    saida[i] = token;
  }
  int posicaoInicial = temLabel ? 2 : 1;
  bool instrucaoDiferente = false;
  for(int i=posicaoInicial; i<tokensLinha.size(); i++){
    string& token = tokensLinha[i];
    Token* tokenObject = pseudo->getToken(i-posicaoInicial+1);
    if(!tokenObject){
      cout << "Instrucao " << pseudo->getToken(0)->token
           << " com campos errados" << endl;
      exit(1);
    }
    string campoPseudo = tokenObject->token;
    if(campoPseudo[0] == '%')
    {
      vector<int> posicoes = pseudo->ProcuraPosicoesCampo(campoPseudo);
      for(unsigned z=0; z<posicoes.size(); z++)
        saida[posicoes[z]] = token;

    } else {
      if(campoPseudo != token)  // Instrucao eh diferente, retornar a original
        return FuncoesGerais::ConcatenaVetor(tokensLinha);
    }
  }
  string stringSaida;
  if(temLabel) {
    stringSaida += tokensLinha[0];
    stringSaida += " ";
  }

  stringSaida += FuncoesGerais::ConcatenaVetor(saida);

  return stringSaida;
}
//------------------------------------------------------------------------
string PreProcessor::ParseLine(string& linha)
{
  char* token;
  token = strtok(FuncoesGerais::StringToChar(linha), "\t ");
  if(token==NULL)
    return "";
```

```
  string primeiroToken(token);
  bool temInstrucao = true;
  bool temLabel = false;
  if(primeiroToken[primeiroToken.length()-1] == ':') {
    temLabel = true;
    temInstrucao = false;
  }

  vector<string> tokensLinha;
  tokensLinha.push_back(token);
  token = strtok(NULL, "\t ");

  while (token)
  {
    temInstrucao = true;
    string stringToken(token);
    vector<string> tokensSeparados = FuncoesGerais::SeparaToken(stringToken);
    for (int i = 0; i < tokensSeparados.size(); i++)
    {
      string& valor = tokensSeparados[i];
      tokensLinha.push_back(valor);
    }
    tokensSeparados.clear();
    token = strtok(NULL, "\t ");
  }
  int posicao = temLabel ? 1 : 0;

  if(temInstrucao) {
    unsigned int nrMacros = macros.size();
    for(unsigned int i = 0; i<nrMacros; i++)
    {
      if(macros[i]->GetPseudoInstrucao()->getToken(0)->token == tokensLinha[posicao])
        return FazTrocaMacro(macros[i], tokensLinha, temLabel);
    }

    unsigned int nrInstrucoes = instrucoes.size();
    for(unsigned int i = 0; i<nrInstrucoes; i++)
    {
      if(instrucoes[i]->getToken(0)->token == tokensLinha[posicao])
        return FazTrocaPseudo(instrucoes[i], tokensLinha, temLabel);
    }

  }

  return linha;
}

//--------------------------------------------------------------------------
string PreProcessor::FazTrocaMacro(
```

```
  Macro* macro,
  vector<string>& tokensLinha,
  bool temLabel
)
{
  string retorno;
  // Ver se a instrucao bate, senão retorna a antiga
  PseudoInstrucao* pseudo = macro->GetPseudoInstrucao();

  int tamanho = temLabel ? pseudo->nrTokens() +1 : pseudo->nrTokens();

  if(tokensLinha.size() != tamanho)
    return FuncoesGerais::ConcatenaVetor(tokensLinha);

  int deslocamento = temLabel ? 1 : 0;

  for(int i=0; i<tokensLinha.size()-deslocamento;i++)
  {
    string campoPseudo = pseudo->getToken(i)->token;
    if(campoPseudo[0] != '%' && tokensLinha[i+deslocamento]!=campoPseudo)
      return FuncoesGerais::ConcatenaVetor(tokensLinha);
  }

  if(temLabel) {
    InstrucaoReal* real = macro->GetInstrucaoReal(0);
    pseudo->setInstrucaoReal(real);
    retorno += FazTroca(pseudo, real,tokensLinha, temLabel);
    retorno += "\n";
    tokensLinha.erase(tokensLinha.begin()); // Apaga o label
  }

  for(int i=deslocamento; i<macro->NumeroInstrucoesReais(); i++)
  {
    InstrucaoReal* real = macro->GetInstrucaoReal(i);
    pseudo->setInstrucaoReal(real);
    retorno += FazTroca(pseudo, real,tokensLinha, false);
    if(i!=macro->NumeroInstrucoesReais()-1)
      retorno += "\n";
  }
  return retorno;

}
//---------------------------------------------------------------------------
```

5.4.11 Arquivo PreProcessor.h

```
//---------------------------------------------------------------------------

#ifndef PreProcessadorH
#define PreProcessadorH
```

```
#include <fstream>
#include <vector>

#include "PseudoInstrucao.h"

class Macro;
class PalavraReservada;

using namespace std;

//---------------------------------------------------------------------------

class PreProcessor {
  public:
    PreProcessor(
      ifstream& arquivoEntrada,
      ofstream& arquivoSaida,
      vector<PseudoInstrucao*>& _instrucoes,
      vector<Macro*>& _macros,
      vector<PalavraReservada*>& _palavrasReservadas,
      vector<string>& _simbolosIgnorar,
      string& _tokenComentario
    );

    void PreProcessaArquivo();


  protected:
    string ParseLine(string& linha);

    string FazTrocaPseudo(
      PseudoInstrucao* pseudo,
      vector<string>& tokensLinha,
      bool temLabel
    );

    string FazTrocaMacro(
      Macro* macro,
      vector<string>& tokensLinha,
      bool temLabel
    );

    string FazTroca(
      PseudoInstrucao* pseudo,
      InstrucaoReal* real,
      vector<string>& tokensLinha,
      bool temLabel
    );
```

```
    string SubstituiPalavrasReservadas(string& linha);

    void SubstituiPalavrasReservadas();

  private:
    ifstream& arquivoEntrada;
    ofstream& arquivoSaida;
    vector<PseudoInstrucao*>& instrucoes;
    vector<Macro*>& macros;
    vector<PalavraReservada*>& palavrasReservadas;
    vector<string>& simbolosIgnorar;
    string& tokenComentario;
};

#endif
```

5.4.12 Arquivo PseudoInstrucao.cpp

```
#include "PseudoInstrucao.h"
//---------------------------------------------------------------------------
void PseudoInstrucao::setInstrucaoReal(InstrucaoReal* _instrucaoReal)
{
  instrucaoReal = _instrucaoReal;
}
//---------------------------------------------------------------------------
InstrucaoReal* PseudoInstrucao::getInstrucaoReal()
{
  return instrucaoReal;
}

//---------------------------------------------------------------------------
vector<int> PseudoInstrucao::ProcuraPosicoesCampo(string& campo)
{
  vector<int> retorno;
  for(int i=0; i<instrucaoReal->nrTokens(); i++){
    if(campo == instrucaoReal->getToken(i)->token)
      retorno.push_back(i);
  }
  return retorno;
}
//---------------------------------------------------------------------------
```

5.4.13 Arquivo PseudoInstrucao.h

```
//---------------------------------------------------------------------------

#ifndef PseudoInstrucaoH
#define PseudoInstrucaoH
```

```
#include "InstrucaoReal.h"
#include <vector>

//------------------------------------------------------------------------
class PseudoInstrucao : public InstrucaoReal
{
  public:
    void setInstrucaoReal(InstrucaoReal* _instrucaoReal);
    InstrucaoReal* getInstrucaoReal();
    vector<int> ProcuraPosicoesCampo(string& campo);

  protected:
    InstrucaoReal* instrucaoReal;

};

#endif
```

5.4.14 Arquivo Semantico.cpp

```
#include "Semantico.h"

#include <iostream>
#include <fstream>

#include "Sintatico.h"
#include "Constants.h"
#include "Lexico.h"
#include "PseudoInstrucao.h"
#include "PreProcessor.h"
#include "Macro.h"
#include "PalavraReservada.h"


namespace PreProcessador {

Semantico::Semantico()
{
  pseudoAtual = NULL;
  instrucaoRealAtual = NULL;
  macroAtual = NULL;
  lendoMacro = false;
  criandoArquivoAddressGenerator = false;
}

void Semantico::executeAction(int action, const Token *token) throw (SemanticError )
{
  string tokenLido = token->getLexeme();
```

```
if(criandoArquivoAddressGenerator){ // Só interessa as ações 7 e 8
  if(action != 7 && action != 8)
    return;
} else {
  if(action == 7 || action == 8)
    return;
}

switch(action)
{
  case 1:
    pseudoAtual = new PseudoInstrucao();
    pseudoAtual->AddToken(tokenLido);
    if(lendoMacro)
      macroAtual->SetPseudoInstrucao(pseudoAtual);

  break;

  case 2: // concatenar % (campo)
    pseudoAtual->AddToken("%"+tokenLido);
  break;

  case 3:
    pseudoAtual->AddToken(tokenLido);
  break;

  case 4:
    instrucaoRealAtual->SetValorFixo(tokenLido);   //??????
  break;

  case 5:
    if(!lendoMacro)
      pseudoInstrucoes.push_back(pseudoAtual);
    instrucaoRealAtual = new InstrucaoReal();
    instrucaoRealAtual->AddToken(tokenLido);
    if(lendoMacro)
      macroAtual->AdicionaInstrucaoReal(instrucaoRealAtual);
    else
      pseudoAtual->setInstrucaoReal(instrucaoRealAtual);

  break;

  case 6:
    lendoMacro = true;
    macroAtual = new Macro();

  break;

  case 7:
```

```
    end = new Enderecos();
    end->setInstrucao(tokenLido);

 break;

 case 8:
 {
   string tokenSemAspas = tokenLido.substr(1,tokenLido.length()-2);
   end->setFormula(tokenSemAspas);
   funcoes.addFuncao(end);
 }
 break;

 case 9:
   macros.push_back(macroAtual);
   macroAtual = NULL;
 break;

 case 10:
 {
   instrucaoRealAtual->AddToken("%"+tokenLido);
 }
 break;

 case 11:
   instrucaoRealAtual->AddToken(tokenLido);
 break;

 case 12:
   palavraReservadaAtual = new PalavraReservada();
   palavraReservadaAtual->setPalavraReservada(tokenLido);
 break;

 case 13:
   palavraReservadaAtual->setValor(tokenLido);
   if(temPalavraReservada(palavraReservadaAtual)){
     throw SemanticError("Palavra reservada repetida: " +
        palavraReservadaAtual->getPalavraReservada(),
           token->getPosition());
     exit(0);
   }
   palavrasReservadas.push_back(palavraReservadaAtual);
 break;

//símbolo do comentário
 case 14:
   setCommentSymbol(tokenLido.substr(1,tokenLido.length()-2));
 break;
```

```
    case 15:
    if (getCommentSymbol() != tokenLido)
        IgnoreSymbols.push_back(tokenLido.substr(1,tokenLido.length()-2));

    break;

  }
  tokenAnterior = tokenLido;
}


bool Semantico::temPalavraReservada(PalavraReservada* palavra)
{

  for(unsigned int x=0; x<palavrasReservadas.size();x++)
    if(palavrasReservadas[x]->getPalavraReservada() == palavra->getPalavraReservada())
      return true;
  return false;

}

vector<PseudoInstrucao*>& Semantico::getPseudoInstrucoes()
{
  return pseudoInstrucoes;
}


vector<Macro*>& Semantico::getMacros()
{
  return macros;
}

vector<PalavraReservada*> Semantico::getPalavrasReservadas()
{
  return palavrasReservadas;
}


void Semantico::setCommentSymbol(string symbol){
  CommentSymbol = symbol;
}

string Semantico::getCommentSymbol(){
  return CommentSymbol;
}

vector<string>& Semantico::getIgnoreSymbols()
{
```

```
  return IgnoreSymbols;
}

} //namespace PreProcessador
```

5.4.15 Arquivo Semantico.h

```
#ifndef SEMANTICO_H
#define SEMANTICO_H

#include "Token.h"
#include "Funcoes.h"
#include "SemanticError.h"
#include <string>
#include <vector>
using namespace std;

class PseudoInstrucao;
class InstrucaoReal;
class Macro;
class PalavraReservada;

namespace PreProcessador {

class Semantico
{
public:
    Semantico();
    void executeAction(int action, const Token *token) throw (SemanticError );
    vector<PseudoInstrucao*>& getPseudoInstrucoes();
    vector<Macro*>& getMacros();
    vector<PalavraReservada*> getPalavrasReservadas();
    void setCommentSymbol(string symbol);
    string getCommentSymbol();
     Funcoes& getFuncoes(){return funcoes;}
    void setCriarArquivoAddressGenerator(bool valor){
      criandoArquivoAddressGenerator = valor;
    }
    vector<string>& getIgnoreSymbols();

protected:
    bool temPalavraReservada(PalavraReservada* palavra);

private:
  PseudoInstrucao* pseudoAtual;
  InstrucaoReal* instrucaoRealAtual;
  PalavraReservada* palavraReservadaAtual;
  Macro* macroAtual;
  string tokenAnterior;
  vector<PseudoInstrucao*> pseudoInstrucoes;
```

```
  vector<PalavraReservada*> palavrasReservadas;
  vector<Macro*> macros;
  Enderecos* end;
  Funcoes funcoes;
  bool lendoMacro;
  bool criandoArquivoAddressGenerator;
  vector<string> IgnoreSymbols;
  string CommentSymbol;


};

} //namespace PreProcessador

#endif
```

5.4.16 Arquivo TratadorDeLabels.cpp

```
#include "TratadorDeLabels.h"
//---------------------------------------------------------------------------
#include "AddressGenerator.h"
#include "Label.h"
#include "Instrucao.h"
#include "ListaInstrucoes.h"
#include "FuncoesGerais.h"


#include <string>
#include <sstream>

using namespace std;

//---------------------------------------------------------------------

char* StringToChar(string& str)
{
  int tamanho = str.length();
  char* retorno = new char[tamanho+1];
  for(int i=0; i<tamanho; i++)
    retorno[i] = str[i];
  retorno[tamanho] = 0;

  return retorno;
}

//---------------------------------------------------------------------

TratadorDeLabels::TratadorDeLabels(ListaInstrucoes* lista, PreProcessor* pp)
{
  listaInstrucoes = lista;
```

```
  preProcessador = pp;

}

//---------------------------------------------------------------------

void TratadorDeLabels::ResolveLabels(string nomeArquivoEntrada, ofstream& arquivoSaida)
{

  ifstream arquivoEntrada(nomeArquivoEntrada.c_str());
  MontaTabelaLabels(arquivoEntrada);
  arquivoEntrada.close();

  ifstream arquivoEntrada3(nomeArquivoEntrada.c_str());
  ResolveInstrucoes(arquivoEntrada3, arquivoSaida);
  arquivoEntrada3.close();

}

//---------------------------------------------------------------------

void TratadorDeLabels::ResolveInstrucoes(
  ifstream& arquivo,
  ofstream& arquivoSaida
)
{
  char linha[1024];
  int contadorEndereco=0;

  vector<Instrucao>& lista = listaInstrucoes->getListaInstr();
  int linhaAtual = 0;

  arquivo.getline(linha, sizeof(linha));
  while (!arquivo.eof())
  {
    linhaAtual++;
    string stringLinha(linha);
    char* token;
    token = strtok(StringToChar(stringLinha), "\t ");
    bool soLabel = false;
    bool temLabel = false;
    if(token!=NULL) {
      string mnemonico(token);
      // O 1o token não eh um mnemonico e sim um label, entao pegar o segundo token
      if(mnemonico[mnemonico.length()-1] == ':')
      {
        temLabel = true;
        token = strtok(NULL, "\t ");
        if(token)
```

```
    mnemonico = token;
  else
    soLabel = true;
}
if(!soLabel) {
  Instrucao* instrucao = ProcuraInstrucao(mnemonico);
  if(!instrucao){
    cout << "Erro linha: " << linhaAtual << endl;
    cout << "Instrucao " << mnemonico << " nao existe" << endl;
    exit(0);
  }
  vector<string> tokensSeparados =
      FuncoesGerais::getTokensSeparados(stringLinha);
  if(instrucao->getTipoBranch() != Instrucao::NENHUM){
    int posicao = instrucao->getPosicaoCampoLabel();
    int contador = 0;
    int posicaoTokenLabel=-1;

    int incrementoLabel = temLabel ? 2 : 0;
    string& nomeLabel = tokensSeparados[
                          posicao+incrementoLabel
                        ];
    int posicaoLabel = BuscaLabel(nomeLabel);
    if(posicaoLabel == -1){
      cout << "Erro linha: " << linhaAtual << endl;
      cout << "Label " << nomeLabel << " nao existente";
      exit(0);
    }
    Label* label = listaLabels[posicaoLabel];
    int endereco = label->GetEndereco();
    int deslocamento = endereco - contadorEndereco;
    deslocamento /= 8;
    int resultado;
    if(instrucao->getTipoBranch() == Instrucao::RELATIVO) {
      resultado = AddressGenerator::calculaEndereco(
                      mnemonico,deslocamento
                  );
    } else {
      resultado = AddressGenerator::calculaEndereco(
                      mnemonico,endereco/8
                  );
    }
    stringstream buffer;
    buffer << resultado;
    tokensSeparados[posicao+incrementoLabel] = buffer.str();
  }
  if(temLabel){
    // Retirar o nome do label
    tokensSeparados.erase(tokensSeparados.begin());
```

```
          // Retirar o ':'
          tokensSeparados.erase(tokensSeparados.begin());
        }
        contadorEndereco += instrucao->getTamanhoInstrucao();
        arquivoSaida << FuncoesGerais::ConcatenaVetor(
                                tokensSeparados) << endl;
      }
    }
    arquivo.getline(linha, sizeof(linha));
  }
}
//---------------------------------------------------------------------------
void TratadorDeLabels::MontaTabelaLabels(ifstream& arquivo)
{
  char linha[1024];
  int contadorEndereco=0;
  vector<Instrucao>& lista = listaInstrucoes->getListaInstr();

  arquivo.getline(linha, sizeof(linha));
  bool temInstrucaoNaLinha = false;
  while (!arquivo.eof())
  {
    string stringLinha(linha);
    char* token;
    token = strtok(StringToChar(stringLinha), "\t ");
    if(token!=NULL) {
      string stringToken(token);
      if(stringToken[stringToken.length()-1] == ':'){
        if(!TemLabel(stringToken)){
          Label* label = new Label(
            stringToken.substr(0,stringToken.length()-1));
          listaLabels.push_back(label);
          label->SetEndereco(contadorEndereco);
        }
        else {
          cout << "ERRO - Label " << stringToken
               << " duplicado"<<endl;
          exit(0);
        }
        // Proximo token sera o mnemonico
        token = strtok(NULL, "\t ");
        if(token){
          stringToken = token;
          temInstrucaoNaLinha = true;
        } else {
          temInstrucaoNaLinha = false;
        }
      } else {
        temInstrucaoNaLinha = true;
```

```
      }
      if(temInstrucaoNaLinha)
      {
        Instrucao* instrucao = ProcuraInstrucao(stringToken);
        if(!instrucao)
        {
          cout << "Instrucao "<< stringToken << " não existente" << endl;
          exit(0);
        } else {
          contadorEndereco += instrucao->getTamanhoInstrucao();
        }
      }
    }
    arquivo.getline(linha, sizeof(linha));
  }

}

//---------------------------------------------------------------------------
Instrucao* TratadorDeLabels::ProcuraInstrucao(string& mnemonico)
{
 vector<Instrucao>& lista = listaInstrucoes->getListaInstr();

  for(int i=0; i<lista.size();i++){
    if(lista[i].getMnemonico() == mnemonico)
      return &lista[i];
  }
  return NULL;
}
//---------------------------------------------------------------------------
bool TratadorDeLabels::TemLabel(string& stringToken)
{
  for(int i=0; i<listaLabels.size(); i++)
  {
    Label* labelAtual = listaLabels[i];
    if(labelAtual->GetNome() == stringToken)
      return true;
  }
  return false;
}
//---------------------------------------------------------------------------

int TratadorDeLabels::BuscaLabel(string& label)
{
  for(int i=0; i<listaLabels.size(); i++)
  {
    Label* labelAtual = listaLabels[i];
    if(labelAtual->GetNome() == label)
      return i;
```

```
  }

 return -1;

}
//---------------------------------------------------------------
```

5.4.17 Arquivo TratadorDeLabels.h

```
//---------------------------------------------------------------------------

#ifndef TratadorDeLabelsH
#define TratadorDeLabelsH

#include <string>
#include <vector>
#include <fstream>
#include <iostream>

#include "PreProcessor.h"

using namespace std;

class Label;
class Instrucao;
class ListaInstrucoes;

class TratadorDeLabels {

  public:
    TratadorDeLabels(ListaInstrucoes* lista, PreProcessor* pp);
    void ResolveLabels(string nomeArquivoEntrada, ofstream& arquivoSaida);

  protected:
    void MontaTabelaLabels(ifstream& arquivo);
    void ResolveInstrucoes(ifstream& arquivo, ofstream& arquivoSaida);
    bool TemLabel(string& stringToken);
    int  BuscaLabel(string& label);
    Instrucao* ProcuraInstrucao(string& mnemonico);

  private:
    vector<Label*> listaLabels;
    ListaInstrucoes* listaInstrucoes;
    PreProcessor* preProcessador;
};


//---------------------------------------------------------------------------
#endif
```

5.5 Anexo IV - Gramática do ArchC

```
#Options
GenerateScanner = true
GenerateParser = true
Language = C++
ScannerName = Lexico
ParserName = Sintatico
SemanticName = Semantico
ScannerCaseSensitive = false
ScannerTable = Full
Input = Stream
Parser = LL
#RegularDefinitions
L : [A-Za-z]
D : [0-9]
WS : [\ \t\n\r]
COMMENT : "/*"([^"*"] | ("*"+ [^"*"/]))* "*"+/ | (/ (/ .*))

#Tokens
"{"
"}"
"["
"]"
"("
")"
";"
","
"="
"-"
"+"
"*"
"/"
"%"
"<"
">"
"<="
">="
"=="
"!="
"!"
"&&"
"||"
":"
","
"."

ID: {L} ({L} | {D} | _)*
NUM: {D}+
HEXA: 0x ({D} | A | B | C | D | E | F | a | b | c | d | e | f)+
```

```
ASPA: \"
ASSEMBLY: \(\"(.)+\"\)

AC_ISA = ID : "AC_ISA"
ISA_CTOR = ID : "ISA_CTOR"
ac_format = ID : "ac_format"
ac_instr = ID : "ac_instr"
ac_addr_mode_R = ID : "ac_addr_mode_R"
ac_addr_mode_A = ID : "ac_addr_mode_A"
set_asm = ID : "set_asm"
set_decoder = ID : "set_decoder"


 : {WS}*
 :! {COMMENT}

#NonTerminals
<ArchC_ISA>
<body>
<format>
<instr>
<addr_mode>
<ISA_ctor>
<id>
<ctor_body>
<decoder>
<campos>
<sinal>
<valor>
<id_campo>
#Grammar
<ArchC_ISA> ::= AC_ISA "(" ID #1 ")" "{" <body> "}" ";" ;

<body> ::= <format> <instr> <addr_mode> <ISA_ctor>;

<format> ::= ac_format ID #3 "=" ASPA <campos> ASPA #7 ";" <format> | î ;

<campos>::= "%" ID #4 ":" NUM #5 <sinal> <campos> |  î;

<sinal>::= ":" ID #6 | î;

<instr> ::= ac_instr "<" ID #8 ">" ID #9 <id> ";" <instr> | î;

<addr_mode> ::= ac_addr_mode_R #15 ID #16 "(" ID #17 ")" <id_campo> ";" <addr_mode> |
                  ac_addr_mode_A #15 ID #16 "(" ID #17")" <id_campo> ";" <addr_mode> | î;
//verificar se o ID entre parenteses é um campo do formato da instrucao correpondente
//verificar se o ID mnemonico já foi declarado na lista de instrucoes
```

```
<id_campo> ::= "," ID #16 "(" ID #17 ")" <id> | î;

<id> ::= "," ID #9 <id> | î;

<ISA_ctor> ::= ISA_CTOR "(" ID #2")" "{" <ctor_body> "}" ";" ;

<ctor_body> ::= ID #10 "." set_asm ASSEMBLY #11";"
                  ID #12"." set_decoder "(" ID #13 "=" <valor> <decoder> ")" ";" <ctor_body> | î;

<valor> ::= HEXA #14 | NUM #14;

<decoder> ::= "," ID #13 "=" <valor> <decoder> | î;

//#1 Guarda ID dizendo que este é nome reservado para o identificador da descricao
//#2 Verifica se ID é o mesmo inserido pela acao 1 (identificador da descricao)
//#3 Verifica se o formato existe; Existe: Erro! Não Existe: Insere e guarda formato_atual
//#4 Verifica se o campo existe; Existe: Erro! Não Existe: Insere campo  e guarda campo_atual
//#5 cria novo objeto campoInstr com o nome campo_atual e tamanho lido
//#6 verifica char; seta sinal para o campo atual se o char for "s"
//#7 cria um novo formato com o nome formato_atual e com os campos lidos e
//#8 verifica se o existe o formato; se sim, guarda(temporário) formato_atual
//#9 verifica se inst existe; se sim, erro,; se não, cria nova instrucao com o nome lido e formato_atual
//#10 Verificar se ID é uma instrucao e ainda não foi setado o seu assemlby.
  // Se é instrucao e não foi setado o seu assembly: guarda "instr_atual", marcar instrucao
  // como tendo assemlby setado
  // Se não é instrucao ou já foi setado o assembly: erro
//#11 setar o assembly da instr_atual para o ASSEMBLY lido

//#12 Verificar se ID é uma instrucao e ainda não foi setado o seu decoder.
  // Se é instrucao e não foi setado o seu decoder: guarda "instr_atual", marcar instrucao como
  // tendo decoder setado
  // Se não é instrucao ou já foi setado o decoder: erro


//#13 verifica se o campo lido faz parte do formato da instr_atual e se seu valor não foi setado
  //se o campo lido faz parte do formato da instr_atual e o valor não foi setado, seta o valor;
  //se o campo não faz parte ou valor já setado, erro.
//#14 seta o valor do campo atual para o valor hexa lido
//#15 Seta uma flag dizendo se é relativo ou absoluto
//#16 Armazena o nome da instrução e seta como sendo de branch
//#17 Seta o campo lido como sendo de endereço
```

5.6 Anexo IV - Gramática do Pré-Processador

```
#Options
GenerateScanner = true
GenerateParser = true
Language = C++
```

```
ScannerName = Lexico
ParserName = Sintatico
SemanticName = Semantico
Package = PreProcessador
ScannerCaseSensitive = false
ScannerTable = Full
Input = Stream
Parser = LL
#RegularDefinitions
L : [A-Za-z]
D : [0-9]
WS : [\ \t\n\r]
COMMENT : "/*"([^"*"] | ("*"+ [^"*"/]))* "*"+/ | (/ (/ .*))

#Tokens
","
"{"
"}"
";"
"="
"%"
":"

ID: {L} ({L} | {D} | _)*
NUM: {D}+
HEXA: 0x ({D} | A | B | C | D | E | F | a | b | c | d | e | f)+
LETTER: {L}
SPECIAL_CHAR: [^ LETTER NUM % \" = \ \n\t\r\{\}:]
QUOTE_STR: \"( [^ ;] )*\"
SEPARATOR: ::

//AC_ISA = ID : "AC_ISA"
AC_PRE_PROCESSOR = ID : "AC_PRE_PROCESSOR"
AC_RESERVED_WORDS = ID : "AC_RESERVED_WORDS"
AC_PSEUDO_INSTRUCTIONS = ID : "AC_PSEUDO_INSTRUCTIONS"
AC_MACRO = ID : "AC_MACRO"
AC_ADDRESS_GENERATOR = ID : "AC_ADDRESS_GENERATOR"
COMMENT_SYMBOL = ID : "COMMENT_SYMBOL"
IGNORE_SYMBOLS = ID : "IGNORE_SYMBOLS"

 : {WS}*
 :! {COMMENT}

#NonTerminals
<GRAM_PP>
<body>
<AC_PSEUDO_INSTRUCTIONS>
<pseudo_instr_list>
<direct_map>
```

```
<macro>
<instr_list>
<native_instr>
<fields_native>
<fixed_value>
<value>
<pseudo_instr>
<fields_pseudo>
<AC_ADDR_GENERATOR>
<operations>
<AC_RESERVED_WORDS>
<reserved_word_list>
<comment>
<ignoresymbols>
<symbols>
#Grammar
<GRAM_PP> ::= AC_PRE_PROCESSOR "{" <body> "}" ";" ;

<body> ::= <AC_RESERVED_WORDS> <AC_PSEUDO_INSTRUCTIONS> <AC_ADDR_GENERATOR>;

<AC_RESERVED_WORDS>::= AC_RESERVED_WORDS  "{" <comment> <ignoresymbols> <reserved_word_list> "}" | î;
<comment> ::= COMMENT_SYMBOL "=" QUOTE_STR #14 ";" | î;
<ignoresymbols>::= IGNORE_SYMBOLS "=" QUOTE_STR #15 <symbols> ";" | î;
<symbols>::= "," QUOTE_STR #15 <symbols> | î;

<reserved_word_list>::= ID #12 "=" HEXA #13 ";" <reserved_word_list> | î;

<AC_PSEUDO_INSTRUCTIONS> ::= AC_PSEUDO_INSTRUCTIONS  "{" <pseudo_instr_list> <macro> "}" | î;

<pseudo_instr_list> ::= <direct_map> <pseudo_instr_list> | î ;

<direct_map> ::= <pseudo_instr>  SEPARATOR <native_instr> ";" ;

<macro> ::= AC_MACRO #6 <pseudo_instr> SEPARATOR "{" <instr_list> "}"#9 <macro> | î ;

<instr_list> ::= <native_instr> ";" <instr_list> | î ;

<native_instr> ::= ID #5 <fields_native> ;

<fields_native> ::= "%" ID #10 <fixed_value> <fields_native> |
                        ID #11 <fields_native> |
                        "{" #11 <fields_native> |
                        "}" #11 <fields_native> |
                        ":" #11 <fields_native> |
                        SPECIAL_CHAR #11 <fields_native> | î ;

<fixed_value> ::= "=" <value> | î;

<value> ::= NUM #4 | HEXA #4 ;
```

```
<pseudo_instr> ::= ID #1 <fields_pseudo> ;

<fields_pseudo> ::= "%" ID #2 <fields_pseudo> |
                        ID #3 <fields_pseudo> |
                        "{" #3 <fields_pseudo> |
                        "}" #3 <fields_pseudo> |
                        ":" #3 <fields_pseudo> |
                        SPECIAL_CHAR #3 <fields_pseudo> | î ;

//Calculo dos enderecos
<AC_ADDR_GENERATOR>::= AC_ADDRESS_GENERATOR "{" <operations> "}" | î;
<operations> ::= ID #7 ":" QUOTE_STR #8 ";" <operations> | î ;
```

5.7 Anexo V - Artigo