MODELAGEM DE ARQUITETURAS VLIW USANDO UMA ADL

Leonardo Taglietti, Flávio Rech Wagner

Universidade Federal do Rio Grande do Sul

{ltaglietti,flavio}@inf.ufrgs.br

ABSTRACT

O presente trabalho aborda a modelagem de processadores VLIW através de uma Linguagem de Descrição de Arquiteturas (ADL), a partir da qual é gerado automaticamente um modelo executável VLIW do processador descrito. Uma abordagem baseada em modelo seqüencial foi publicada e apresentada no *XI Workshop Iberchip/2005*. Entretanto, o objetivo principal do modelo VLIW aqui proposto é usufruir a vantagem de execução paralela de instruções visando reduzir o número de instruções executadas pelo processador e, conseqüentemente, o tempo de execução dos programas. Os resultados obtidos com os experimentos realizados mostram que, com o modelo VLIW, obtém-se ganhos que variam de 10 a 30 % em relação ao modelo seqüencial.

1. INTRODUÇÃO

Diversas soluções no nível de organização e de arquitetura de processadores surgiram nas últimas décadas, com o objetivo de aumentar o desempenho dos mesmos. O uso de *pipelines* permite a sobreposição no tempo [1]. Nas arquiteturas superescalares, a extração de paralelismo é realizada através de *hardware* especializado, cujos mecanismos geralmente ocupam uma grande área no circuito. Já em arquiteturas VLIW (*Very Long Instruction Word*) explora-se o paralelismo no nível de instruções (ILP) em tempo de compilação, liberando o *hardware* desta tarefa [2]. As dificuldades de obtenção de paralelismo são conseqüências naturais do modelo seqüencial de processadores projetados inicialmente na década de 1940.

Os processadores podem ser descritos através de *Hardware Description Languages* (HDLs), tais como VHDL e Verilog. Entretanto, estas linguagens permitem apenas a descrição de componentes no nível RT (*register-transfer*), aumentando o tempo de projeto devido à quantidade de detalhes exigidos em descrições de baixo nível de abstração.

O projeto no nível de sistemas (*system-level design*) representa o aumento no nível de abstração para descrição de sistemas e vem tornando-se um padrão na indústria [3]. Este tipo de projeto é suportado por SystemC [4], uma extensão das bibliotecas padrão da linguagem C++, através da qual é possível descrever *timing*, módulos, paralelismo do *hardware*, etc. A união destes aspectos torna SystemC uma das linguagens mais promissoras para a modelagem de sistemas embarcados [3].

No projeto de uma arquitetura, a inexistência de *software* básico (tais como compilador, montador, ligador, etc.) torna mandatória a geração automática destas ferramentas, principalmente diante da pressão do *time-to-market*. As Linguagens de Descrição de Arquiteturas (ADLs) surgiram com intuito de facilitar a especificação do conjunto de instruções de processadores. Entre as ADLs mais conhecidas estão LISA, EXPRESSION e ArchC [5].

Além de estarem inseridas no contexto de *co-design* (projeto paralelo de *hardware* e *software*), as ADLs facilitam a geração automática de simuladores, montadores, compiladores,etc, já que a utilização direta de SystemC como ponto de partida inviabiliza o processo de geração automática de ferramentas, devido aos inúmeros estilos possíveis de descrição de modelos.

Este artigo aborda a modelagem de processadores VLIW a partir da ADL ArchC [5], através da qual é gerado automaticamente um modelo executável do processador VLIW descrito. Para estudo de caso, foi adotado um modelo do microcontrolador PIC 16F84, disponível em repositório público [5]. A abordagem aqui adotada é o aproveitamento da organização original do microcontrolador em termos de unidades funcionais, focando na extensão do conjunto de instruções. Os modelos VLIW e funcional não possuem *pipeline*. O trabalho que detalha o modelo funcional do PIC 16F84 foi publicado no *XI Workshop Iberchip/2005* [6].

O objetivo deste trabalho é usufruir a execução paralela de instruções para reduzir o número de instruções executadas pelo processador, bem como o tempo de execução dos programas. Os resultados obtidos com os experimentos evidenciam ganhos que variam de 10 a 30 %

em relação ao modelo seqüencial existente.

O restante deste artigo é assim organizado: a Seção 2 descreve os trabalhos correlatos. A Seção 3 mostra o desenvolvimento e extensão do modelo, bem como as restrições da arquitetura na obtenção de paralelismo. Os procedimentos experimentais e os resultados obtidos são mostrados na Seção 4. Na Seção 5, são apresentadas as conclusões e perspectivas de trabalhos futuros.

2. TRABALHOS CORRELATOS

Algumas ADLs difundidas são LISA, LPDP, EXPRESSION, ISDL e ArchC [5]. Este trabalho utiliza ArchC, motivado por ser a única ADL que gera, automaticamente, simuladores (modelos executáveis) escritos em SystemC [4]. Além disso, ArchC foi adotada na descrição do modelo convencional do PIC 16F84, objeto de comparação direta neste trabalho.

ArchC suporta simulação, co-verificação e geração automática de montadores. Estão em desenvolvimento os geradores de compiladores e ligadores (*linkers*). Esta ADL é licenciada sob a GNU *Lesser General Public License* (LGPL).

Alguns processadores modelados em ArchC estão disponíveis no repositório da ADL [5]. Entre eles, o PIC 16F84 e o i8051 foram desenvolvidos em cooperação com outros grupos de pesquisa. Um modelo de processador digital de sinais (DSP) está em desenvolvimento [5]. Trata-se de uma modelagem de uma arquitetura originalmente VLIW, na qual o comportamento das instruções é adaptado fielmente com a arquitetura. Já no modelo aqui proposto, somente são feitas as devidas adaptações para que sejam chamados os comportamentos já existentes no modelo funcional, de acordo com a instrução VLIW decodificada.

3. ESTUDO DE CASO: PIC 16F84

O PIC 16F84 é um microcontrolador da família RISC, baseado em Acumulador, e adota arquitetura *Harvard*. Sua versão original possui 35 instruções, distribuídas em 4 diferentes formatos. Com exceção das instruções de desvio (que gastam dois ciclos de *clock*), todas as outras instruções consomem apenas um ciclo de relógio.

3.1. O desenvolvimento do modelo

Um modelo ArchC é composto da descrição do conjunto de instruções da arquitetura (AC_ISA), onde são especificados os formatos, mapeamento de mnemônicos e códigos operacionais de cada instrução. Na descrição da organização da arquitetura (AC_ARCH), descreve-se as memórias, registradores, etc.

O pré-processador ArchC (*acpp*) interpreta estas descrições e gera automaticamente o simulador da arquitetura. Entretanto, os esqueletos dos métodos de cada instrução são gerados parcialmente, permitindo assim que o projetista especifique o comportamento desejado de cada instrução.

3.2. Detalhes de implementação

Os formatos de instrução do modelo convencional do PIC 16F84 [6] são apresentados na Tabela 1.

Tabela 1. Formatos de instrução no PIC 16F84

Formato	*Campo: Tamanho*	*Bits*
Byte	dummy:2 opbyte:6 d:1 f:7	16
Bit	dummy:2 opbit:4 b:3 f:7	16
Literal	dummy:2 oplit:6 k:8	16
Controle	dummy:2 opctrl:3 kaddress:11	16

Originalmente, o PIC tem 14 bits nos formatos de instrução. Entretanto, para fins de implementação e sem prejuízo à modelagem proposta, foi inserido o campo *dummy*, tornando os formatos com tamanho múltiplo de 8.

A Tabela 2 mostra os *templates* de instrução implementados no modelo VLIW.

Tabela 2. *Templates* VLIW

Template	*Formatos envolvidos*
A	Literal e Controle
B	Byte e Bit
C	Byte e Literal
D	Byte e Controle
E	Byte e Byte

Estes *templates* (formatos VLIW) são formados pela união dos formatos originais do PIC. Caso não existam restrições (tais como dependência de dados e de controle), as instruções enquadradas nestes formatos podem ser executadas em paralelo.

Um exemplo de descrição VLIW do PIC-*Long Word* (PIC-LW) é apresentado resumidamente na Figura 1.

Inicialmente, existe a declaração de um formato (*Format_Byte_Literal)*, composto pela união dos formatos *Byte* e *Literal* do modelo original do PIC [6]. A união de dois formatos resulta em 32 bits, sendo este o tamanho da palavra (molécula) VLIW.

A Seção B mostra a associação de uma instrução com um respectivo formato. Já na Seção C (ISA_CTOR), é definida a sintaxe *assembly*, juntamente com os códigos operacionais de cada instrução.

```
AC_ISA(LW){
//Seção A: Descrição dos Formatos de Instrução
ac_format Format_Byte_Literal =
"%dummyFBLWord1:2 %opbyteFBLWord1:6
%dFBLWord1:1 %fFBLWord1:7
%dummyFBLWord2:2 %oplitFBLWord2:6
%kFBLWord2:8";
    ...
//Seção B: Mapeamento de Instruções para
formatos
ac_instr<Format_Byte_Literal> CLRF_MOVLW;
    ...
//Seção C: Mapeamento de mnemônicos e códigos
operacionais
  ISA_CTOR(LW){
   CLRF_MOVLW.set_asm("CLRF %fFBLWord1 MOVLW
   %kFBLWord2");
   CLRF_MOVLW.set_decoder(opbyteFBLWord1=0x01,
   dFBLWord1=1,oplitFBLWord2=0x30);
  };
};
```

Figura 1. Exemplo de descrição ArchC do PIC-LW

3.3. Particularidades do microcontrolador

O PIC 16F84 tem 35 instruções, com as quais é possível executar operações com literais (valores imediatos), manipulações de *bytes,* bits, etc. Operações de desvios incondicionais e chamadas de sub-rotinas são realizadas pelas instruções GOTO e CALL, respectivamente.

O PIC possui um registrador denominado *Status*, o qual é localizado no endereço 03 do banco de registradores. Comandos condicionais (*if-then-else*), operações aritméticas e lógicas tais como incrementos, decrementos, subtrações, rotações, etc., registram o estado da operação da ALU no *Status*.

Os desvios condicionais e controle de iteração de *loops* são realizados através de incrementos e subtrações nos registradores envolvidos, com o posterior acesso ao registrador *Status,* mais precisamente aos bits *carry*, *digit carry* e *zero*. A verificação destes bits na execução das instruções é fundamental para determinar a semântica e o correto fluxo do programa.

3.4. Metodologia para extração de paralelismo

O objetivo inicial na construção do modelo VLIW é buscar trechos de código paralelizáveis. Das 35 instruções existentes, 18 instruções fazem algum acesso ao *Status* (leitura ou escrita) e, portanto, não poderão ser paralelizadas. O *Status* é a estrutura principal a ser observada na busca de paralelismo no PIC.

Outra análise mostra que duas instruções que acessam um mesmo registrador não poderão estar na mesma palavra VLIW. Além disso, nas instruções com dois operandos, um deles estará sempre no acumulador. Então, duas instruções (contendo dois operandos cada) devem estar em diferentes palavras VLIW para garantir a integridade dos dados.

4. RESULTADOS EXPERIMENTAIS

4.1. Configuração experimental

Nos experimentos, foi utilizada a ADL ArchC 1.2 e o compilador *PIC C Compiler*, versão 3.182 [8]. Um montador gerado automaticamente [7] foi usado para realizar a montagem do código *assembly* produzido pelo compilador adotado.

4.2. Procedimentos experimentais

Primeiramente, o código *assembly* de cada *benchmark* foi particionado em blocos básicos [9]. Quando a última instrução do bloco básico for uma instrução de desvio condicional, somente depois da sua execução é que será possível determinar qual o próximo bloco de código que será executado.

Um exemplo de código seqüencial é mostrado na Figura 2, onde as seis primeiras instruções representam um *loop*. A quinta instrução (DECFSZ) testa se o registrador 0x0D possui o valor zero. Se o valor for diferente de zero, o *loop* é reiniciado. Caso o valor seja igual a zero, a sexta instrução (GOTO) é ignorada, e o *loop* é encerrado.

```
…
L1: MOVWF 0x0D
    BTFSC 0x03,2
    BCF 0x03,0
    RLF 0x0C,1
    DECFSZ 0x0D,1
    GOTO L1
    MOVLW 0x00
…
```

Figura 2. Exemplo de *loop*

A análise de código do *loop* mostrado na Figura 2, considerando as restrições do PIC apontadas na Seção 3.3, possibilitou a geração de código correspondente ao modelo VLIW, mostrado na Figura 3.

Address	Word 1	Word 2
...		
30	MOVWF 0x0D	BTFSC 0x03,2
34	BCF 0x03,0	RLF 0x0C,1
38	DECFSZ 0x0D,1	Nop
3C	GOTO 030	Nop
40	MOVLW 0X00	Nop
...		

Figura 3. Código para o PIC-LW

Note que, no endereço 30, só a instrução BTFSC faz acesso ao registrador de *Status* (0x03). Não há conflito de registradores e nem dependência de dados entre elas,

possibilitando assim a paralelização. Já no endereço 34, a instrução RLF afeta o *Status*, mas a instrução BCF não realiza acesso ao mesmo. Novamente, não há dependência de dados e nem qualquer conflito, indicando que ambas podem estar na mesma molécula VLIW. Instruções que não podem ser paralelizadas exigem uma instrução NOP (não-operação) para preencher os bits restantes da palavra.

Note que a paralelização ocorreu dentro de um mesmo bloco básico (endereços 30 à 3C). A instrução do endereço 40 (MOVLW) faz parte de outro bloco básico.

Entre os mecanismos de otimização empregados em compiladores, destaca-se o uso de Grafos de Fluxo de Controle (CFG), que também facilitam a otimização de *loops.* Qualquer paralelização realizada em blocos de um *loop* reduz o número de instruções executadas.

A partir dos blocos básicos, foi construído o CFG de cada *benchmark* (a ser apresentados na Seção 4.3), o qual mostra os caminhos atingíveis a partir de cada bloco. A Figura 4 representa o grafo de fluxo de controle obtido a partir dos blocos básicos do *benchmark int2bin*[10].

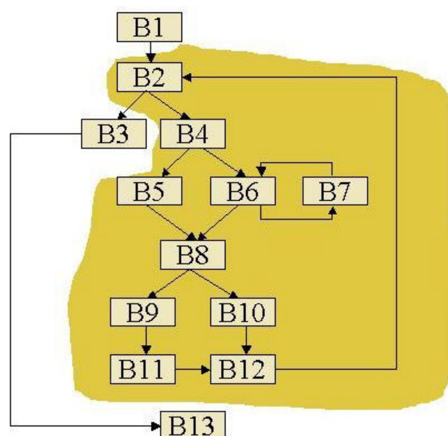

Figura 4. CFG do *benchmark int2bin*

Os nodos representam os blocos e as arestas representam o fluxo entre eles. Como os blocos B1, B3 e B13 não fazem parte de *loop*, suas otimizações causam menor impacto em relação aos outros nodos. A região sombreada representa um *loop*, formado pelos nodos restantes. Note que também há um *loop* interno entre os nodos B6 e B7. A paralelização de instruções em blocos que fazem parte de *loops* causa um maior impacto, pois a cada iteração do laço, serão executadas menos instruções se comparado ao mesmo programa seqüencial.

4.3. Validação do modelo do PIC-LW

Cada instrução VLIW foi testada individualmente, possibilitando realizar somas, subtrações, incrementos, etc., além de fazer a depuração dos bits do registrador *Status*, um dos focos da abordagem apresentada neste trabalho. Testes posteriores foram realizados utilizando pequenos programas com estruturas *if-then-else*, buscando avaliar o comportamento das instruções de desvio.

Os experimentos mais avançados foram feitos com uso de *benchmarks* obtidos no Projeto Dalton [10], os quais realizam operações sobre inteiros, conversões de tipo, e números negativos. Os resultados obtidos para todos os *benchmarks* testados eram esperados, visto que a paralelização de instruções em blocos básicos permitiu a redução do número de instruções executadas.

4.4. Comparação com o modelo funcional do PIC

Esta seção apresenta uma comparação do modelo do PIC-LW com o modelo funcional já certificado. Para cada *benchmark* [10], a Tabela 3 compara o número de instruções executadas nas duas versões do PIC.

Tabela 3. Instruções executadas

Programa	Instruções executadas	
	PIC	PIC-LW
negcnt	150	121
int2bin	256	174
fib	358	325

O número de instruções executadas para o PIC-LW é menor em todos os casos testados. No programa *fib*, a redução aproximada foi de 10 %. No programa *negcnt*, a redução aproxima-se de 20%. O programa *int2bin* apresentou a maior redução: cerca de 32 %. Os dados da Tabela 3 são evidenciados na Figura 5.

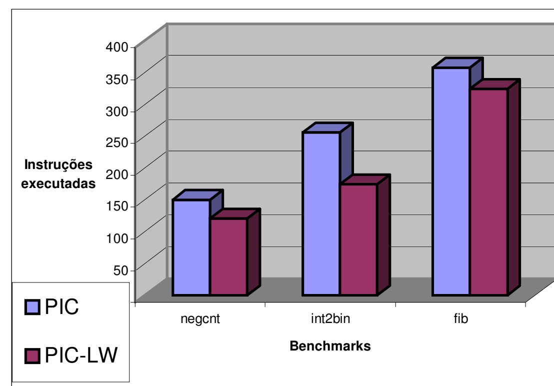

Figura 5. Instruções executadas: PIC x PIC-LW

Para todos os *benchmarks* testados, houve redução no tempo de execução dos *benchmarks*, o qual decresce linearmente com a redução do número de instruções executadas no modelo VLIW.

5. CONCLUSÕES E TRABALHOS FUTUROS

Inicialmente, comprovou-se que a ADL adotada permite o desenvolvimento de modelos VLIW. Entretanto, neste trabalho não foi considerada a duplicação de unidades

funcionais e registradores de predicado, o que poderia exigir pequenas extensões na ADL. Como exemplo de extensão cita-se a possível integração de diferentes recursos arquiteturais com o uso de *threads*. Estas, por sua vez, estão disponíveis na biblioteca SystemC, base da ADL ArchC.

Mudanças na organização original do PIC, tais como a inclusão de outra unidade lógica-aritmética influenciaria na redução do número de instruções executadas, visto que poderiam ser paralelizadas somas, subtrações, etc.

A indisponibilidade de um compilador exigiu grande esforço manual na paralelização de código, totalizando aproximadamente 18,4 horas de trabalho. Portanto, o desenvolvimento de um compilador seria necessário para facilitar os experimentos e a validação de novos modelos. Outra alternativa é a construção de uma ferramenta de conversão de código seqüencial em código VLIW, bem como a extensão do gerador de montadores [7] para amparar tais experimentos.

A partir de *benchmarks*, verificou-se que a molécula de 32 bits é suficiente para o PIC, pois dificilmente três ou mais instruções poderiam ser executadas em paralelo. Os resultados obtidos com os *benchmarks* testados foram suficientes para validar experimentalmente o modelo VLIW, já que a validação funcional do modelo foi realizada em [6]. A eficiência foi comprovada pela redução do número de instruções executadas e a robustez do modelo é garantida pelos *benchmarks* adotados.

Estima-se que a versão VLIW do PIC ocuparia uma maior área no *chip,* com possível maior custo de potência. Assim, o modelo aqui apresentado pode ser estendido para uma versão com precisão de ciclos, subsidiando experimentos precisos envolvendo tempo, potência e área.

6. REFERÊNCIAS

[1] David A. Patterson & John L. Hennessy. *Organização e Projeto de Computadores. A Interface Hardware/Software.* Segunda Edição. LTC Editora.

[2] Wagner, Flávio R. Notas de Aula. CMP 237 – Arquitetura e Organização de Processadores.UFRGS/2005.

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

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

[5] Rigo, S. "The ArchC Architectural Description Language". Disponível em http://www.archc.org. UNICAMP,2004.

[6] Taglietti, L et al. "Modelagem do Microcontrolador PIC 16F84 para projeto baseado no reuso de IPs". XI Workshop Iberchip 2005, pg 303.

[7] Taglietti, L., Carlomagno Filho, J. O., Casarotto, D. C., Furtado, O. J. V., Santos, L. C. V. "*Automatically Retargetable Pre-Processor and Assembler Generation for ASIPs"*. 3rd International IEEE Northeast Workshop on Circuits and Systems (IEEE-NEWCAS'05)

[8] *PCW Compiler:* Custom Computer Services, Inc. http://www.ccsinfo.com/

[9] Aho, A.; Sethi, R.; Ullman, J. D. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1988.

[10] Dalton Project. http://www.cs.ucr.edu/~dalton/