ALAN FISCHER E SILVA
DANILO CESAR LEMES DE PAULA

Ambiente de simulação para sistemas embarcados

Trabalho de Graduação em Bacharelado em Ciência da Computação, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Roberto Hexsel

CURITIBA

2007

ALAN FISCHER E SILVA
DANILO CESAR LEMES DE PAULA

Ambiente de simulação para sistemas embarcados

Trabalho de Graduação em Bacharelado em Ciência da Computação, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Roberto Hexsel

CURITIBA

2007

Sumário

Agradecimentos

Agradecemos primeiramente, o professor doutor Roberto André Hexsel pela orientação no decorrer das pesquisas e do trabalho, bem como pela paciência durante toda a orientação. À Andreia Barbiero e ao Bruno H. Hjort, que já trabalharam com os simuladores escolhidos e que auxiliaram-nos em momentos de dúvidas e também ao LARSIS por ter permitido o uso de suas dependências durante o projeto.

Capítulo 1

Resumo

Como visto em disciplinas como Arquitetura de Computadores e Tópicos em Arquitetura de Computadores, atualmente a produção de software para Desktops se baseia em torno de 10% de todo software novo produzido enquanto que a produção de software para sistemas embarcados se baseia em torno de 70% de todo o software novo produzido. Baseado nestas informações e levando em conta o mercado cada vez mais competitivo é extremamente importante que ferramentas adequadas auxiliem os projetistas na escolha do hardware mais adequado para uma determinada aplicação e que o desenvolvimento do código possa ser iniciado antes mesmo do término do hardware. Também levando em conta o constante crescimento que o software livre vem apresentando em quase todos os segmentos da tecnologia da informação seria extremamente interessante disponibilizar alternativas de código aberto e gratuitas tendo em vista que muitas vezes o projetista só possuí a opção do software proprietário, o qual muitas vezes requer licenças , o que encarece o desenvolvimento de dispositivos e aplicações.

Neste trabalho descrevemos a migração completa do modelo ATMega8515 da versão 1.6 do ArchC para a versão 2.0 do mesmo, no qual pudemos comparar o modelo funcional com o modelo multi ciclo, bem como a migração parcial da versão 1.6 do ArchC para a versão 2.0 dos modelos Rabbit R3000 e Motorola DSP56F827. Como parte da pesquisa realizada também descrevemos uma ferramenta importantíssima do ArchC que permite gerar montadores a partir do código de descrição do processador e que a partir desta ferramenta pudemos gerar um montador baseado no GAS (Gnu Assembler) para o modelo Motorola DSP56F827. A pesquisa sobre esta ferramenta de geração de montadores foi demasiadamente trabalhosa tendo em vista que tal ferramenta esta muito pouco documentada no site do ArchC o que exigiu um bom entendimento do funcionamento da ferramenta antes de proceder com a geração do montador.

Capítulo 2

Abstract

As seen in previous subjects like Computer Architecture Topics and Computer Architecture, nowadays the software production for Desktops are based in more or less 10% from all new produced software while the software production for embedded systems are 70% from all new produced software. Regarding these information and the fact about the market being day by day more competitive, is extremely important that adequate tools help the developers choosing the most advanced hardware for a specific application and that the code development might began before finishing the hardware. Taking care also the continuous growth that the free software is presenting in almost all segments from IT - Information Technology would be extremely interesting provide open source and free alternatives because many times a developer only have the proprietary software choice which in the most cases requires licenses making the development cost of the solutions rises.

In this paper we describe the complete migration from ATMega8515 from ArchC 1.6 version to ArchC 2.0 version where we could compare the mono cycle model against the multi cycle model, as well the partial migration from ArchC 1.6 version to ArchC 2.0 version of the Rabbit R3000 and Motorola DSP56F827 models. As part of the research we also describe an important ArchC tool that permits generate assemblers from the description code of the processor model and from this work we could generate an assembler based in GAS (GNU Assembler) for Motorola DSP56F827 model. The research about this assembler generation tool was very hard taking care that this tool does not have very documentation in ArchC site what demanded a good understanding of the tool process before proceeding with the assembler generation.

Capítulo 3

Introdução

Sistemas embarcados são sistemas que realizam uma tarefa específica, geralmente em dispositivos com limitações de memória, de processamento, de armazenamento e principalmente de energia. Um sistema embarcado deve ser projetado para ser otimizado de acordo com os recursos disponíveis, bem como, fazer um bom uso dos mesmos.

A simulação de microprocessadores permite obter resultados próximos da realidade sem precisar trabalhar diretamente no dispositivo, o que tende a consumir mais tempo e recursos financeiros. Os simuladores permitem prever com razoável precisão, no início do ciclo de desenvolvimento se determinado processador tem desempenho suficiente para a aplicação, qual será o tamanho do código compilado e qual a capacidade mínima da memória de dados. Possuindo uma estimativa próxima da realidade é provável que a escolha do microprocessador ou microcontrolador tenha grandes chances de estar correta. Um dos problemas enfrentados atualmente é que os fabricantes não fornecem ferramentas adequadas para este tipo de comparação e quando elas existem, são utilizadas como software proprietário e de alto custo, o que encarece o desenvolvimento.

Uma das maneiras de realizar a simulação de microprocessadores em ambiente GNU/Linux é utilizar a ferramenta ArchC, que é uma ferramenta de código aberto para descrever arquiteturas de microprocessadores de forma eficiente e rápida e que a partir da descrição, gera automaticamente o simulador capaz de estimar por exemplo o tempo de execução, o número de ciclos e o número de acessos a memória.

O ArchC é baseado no SystemC, uma linguagem de descrição de hardwares que recebe forte suporte de muitas fabricantes de processadores. Com o ArchC, além de descrever diferentes arquiteturas de microprocessadores, é possível analisar os resultados obtidos na execução dos

programas. Outra funcionalidade desta ferramenta é a possibilidade de construir montadores baseado nas descrições da arquitetura.

Os simuladores utilizados foram implementados na versão 1.6 do Archc pela Andréia Barbiero e foram escolhidos por se tratarem de microcontroladores amplamente utilizados no mercado. É importante ressaltar que os simuladores citados podem ser facilmente expandidos para aceitar a simulação de outros processadores das três famílias, já que em muitas vezes o conjunto de instruções é bastante parecido e será necessário apenas adicionar ou remover algumas instruções.

Outro assunto importante a ser discutido neste trabalho é o fato de que muitas vezes os montadores/compiladores disponibilizados pelas fabricantes de microprocessadores e microcontroladores de uso específico são disponibilizados apenas em formato de software proprietário ou então possuem versões apenas para o sistema operacional Microsoft Windows. De nada adiantaria ter um simulador adequado se não houver a possibilidade de testar os programas à fim de realizar comparações e com isso escolher o melhor ambiente.

Capítulo 4

Ferramentas Utilizadas

4.1 SYSTEMC

O SystemC é uma biblioteca de classes que estende a linguagem C++ utilizada para modelar sistemas e foi desenvolvida em 1999 através de uma cooperação entre a Synopsys, a CoWare/IMEC e a Universidade da Califórnia - Irvine. Hoje, várias instituições, empresas e fabricantes de hardware compõe a Open SystemC Initiative e disponibilizam ferramentas e código fonte na página do projeto. Para baixar o código fonte é bem simples, basta se cadastrar na página do projeto - www.systemc.org. Esta ferramenta vem sendo adotada por inúmeros fabricantes como linguagem padrão de modelagem e simulação. O SystemC possibilita a descrição do comportamento , uma noção de tempo em operações seqüências além de tipos de dados para hardware, estrutura hierárquica e suporte à simulação.

Em SystemC, um sistema pode ser modelado como uma coleção de módulos que contém processos, portas e canais.

- Um processo define o comportamento de um módulo específico além de disponibilizar métodos para expressar concorrência.
- Um canal implementa uma ou mais interfaces, onde uma interface é uma coleção de métodos ou funções
- Uma porta é utilizada por um processo para acessar a interface de um canal.

4.2 ArchC

O ArchC é uma ferramenta de descrição de microprocessadores baseada em SystemC e que permite a geração automática de simuladores funcionais, multi-ciclos e segmentados permitindo medir diretamente grandezas como o número de ciclos, o tempo de execução, o número de instruções executadas, o número de chamadas de Entrada e Saída e o número de acessos que cada instrução faz à memória.

O ArchC utiliza as bibliotecas do SystemC para modelar o hardware a ser simulado, e inclui ferramentas capazes de automatizar o processo de construção do simulador. Para gerar um simulador de um microprocessador com o ArchC é necessário descrever as características básicas da organização do mesmo e descrever a sintaxe e a semântica do conjunto de instruções. Para este processo é necessário descrever estas características em 3 arquivos diferentes:

- modelo.ac - Onde são descritas as características do processador tal como o tamanho da memória de dados, o tamanho da memória de programa, tamanho da palavra e a descrição dos registradores.
- modelo_isa.ac - Onde são descritos os tipos de instruções e o cabeçalho das instruções.
- modelo_isa.cpp - Onde é feita a descrição do comportamento das instruções em código C/C++.

Outra características importantes do ArchC é que o mesmo permite a construção de simuladores Interpretados e Compilados, o que confere uma grande vantagem em relação às demais ferramentas, que geralmente focam em apenas um dos modelos.

Além disso, o ArchC permite:

- Uma simulação detalhada da hierarquia de memória
- Simuladores gerados emularem chamadas ao sistema operacional.
- Geração automática de montadores. Desde a versão 1.5 até a versão 1.6 esta funcionalidade era permitida apenas para arquiteturas com instruções de tamanho fixo, porém a partir da versão 2.0 Beta é permitido conjunto de instruções com tamanhos variáveis.
- Geração de ligadores (A partir da versão 2.0 beta).
- Suporte ao protocolo TLM (Transaction Level Modeling) utilizado para comunicação.

4.3 GNU binutils

A Gnu BinUtils é uma coleção de ferramentas binárias utilizadas para gerar, editar e visualizar informações em arquivos binários. As principais ferramentas são:

- as
- ld

Para este trabalho, foi utilizada a versão GNU-Binutils 2.16, que deverá ter seu código fonte obtido através do endereço ***http://www.gnu.org/binutils***

O conjunto do código fonte do binutils faz-se necessário neste trabalho para, através da ajuda do sistema de geração de montadores do archC, gerar uma ferramenta (no caso o *GNU assembler*, ou *gas*) capaz de transformar código Assembly em código objeto para a arquitetura descrita através dos padrões do archC, descritos no capítulo 8.

4.4 Tavrasm

Tavrasm é um montador para os micro-controles da família Atmel AVR.

Ele usa a sintaxe similar aos AVR-assembler desenvolvidos para MS-DOS.

Mais informações na seção 11.1.4

Capítulo 5

Descrição dos Modelos escolhidos

Neste capítulo, descreveremos as características dos modelos escolhidos, modelos estes que possuem diferenças em termos de recursos, custos e desempenho.

O Atmel Atmega8515 é um processador de 8 bits de dados, com instruções de 16 bits, enquanto o Rabbit é um processador de 8 bits com instruções de tamanho variável. Já o Motorola DSP56F827 é um processador de 16 bits de dados com instruções de tamanho variável.

Além da diferença nos processadores existe uma diferença nos recursos, sendo que o Motorola DSP56F827 possui um processador de 80 Mhz, possuindo memória Flash de 128 Kbytes, memória RAM interna de 8192 Bytes e memória RAM externa de 128 Bytes, o seu custo é de aproximadamente 11 $ e o mesmo possui uma dissipação de potência de 150 mW a 198 mW.

O Atmel Atmega8515 possui um clock de 16 Mhz, memória flash de 8 Kbytes, memória RAM interna de 512 Bytes e memória RAM externa de 64 Kbytes, o seu custo é de aproximadamente 4,85 $ e o mesmo possui uma dissipação de potência de 12mW a 60mW.

O Rabbit possui um clock de 30 Mhz, memória flash de 64 Kbytes (sendo que a memória de dados e de código é compartilhada), memória RAM interna de 64 Kbytes (sendo que a mesma é memória de código e memória de dados) e memória RAM Externa de 1024 Kbytes, o seu custo gira em torno de 12,75 U$ e possui uma dissipação de potência de 22 mW.

Capítulo 6

Migração das arquiteturas e detalhes técnicos

6.1 Migração das arquiteturas do archC 1.6 para ArchC 2.0

Como citado anteriormente, o archC 2.0 permite que usemos uma série de funções extras não presentes nas versões 1.x.

Algumas destas funcionalidades são bastante importantes para quem trabalha com simulações de microprocessadores. O archC 2.0 permite um melhor estruturamento do código do simulador, comunicação externa compatível com o padrão TLM e multi-simulação para sistemas multiprocessados, além de simulação 30x mais rápida.

Outra vantagem de manter o código dos processadores atualizado para as versões mais recentes do archC é o fato de que eles manteriam-se funcionais, passíveis de serem utilizados em produção em qualquer momento. Além de ter seu código revisado e melhorado durante o período de revisões para aproximar seus resultados aos resultados de um modelo real.

6.2 Detalhes técnicos sobre o processo de migração

Basicamente, na versão 2.0 houveram várias modificações na estrutura do "kernel"do archC. Algumas destas modificações implicaram na incompatibilidade de certas abordagens de programação comumente utilizadas na versão 1.6.

Um exemplo disto é que nas versões mais antigas do archC, a organização dos componentes do microprocessador eram representadas por variáveis globais dentro dos arquivos. Ou seja, poderíamos, através de qualquer função dentro do código, acessar e modificar o conteúdo dos

componentes de memória e registradores.

Esta abordagem foi totalmente proibida após a revisão beta3 do archC através do uso de "interfaces". Isto exigiu algumas modificações no código fonte dos simuladores, tais como modificar toda função que se utiliza dos componentes de memória ou registradores, afim de enviar como parâmetro o endereço destes componentes.

Também foi necessário, no código do processador, incluir as novas macros do archC que estam presentes nos arquivos ¡nome do simulador¿_bhv_macros.H e ¡nome do simulador¿_isa-init.cpp. Outro detalhe importante que foi corrigido na versão 2.0 do archC é que os registradores seriam inicializados com "lixo"o que não acontecia nas versões 1.x, onde os mesmos era inicializados com 0. Este detalhe torna a simulação mais próxima do mundo real, porém é importante ressaltar que para efeitos de testes tivemos de zerar todos os registradores do Atmega pois um dos programas de testes fazia o uso de tal recurso, comparando variáveis não inicalizadas com zero.

O trecho de código pode ser identificado abaixo e se comentado faz com que o Atmel funcione da maneira correta onde a instrução RB.write(i,0) escreve o valor 0 em cada registrador indexado por i.:

```
SREG.I = 0;
SREG.T = 0;
SREG.H = 0;
SREG.S = 0;
SREG.V = 0;
SREG.N = 0;
SREG.Z = 0;
SREG.C = 0;
for (int i = 0; i < 32; i++){ RB.write(i,0); }
```

6.3 ATMEL Atmega8515

Como citado anteriormente, uma das arquiteturas migradas para a versão 2.0 do archC foi a arquitetura do microprocessador atmega 8515 da Atmel Corporation.

Obtivemos nesta arquitetura os melhores resultados durante o processo de simulação de instruções. Todos as simulações funcionaram de forma idêntica às simulações realizadas com as versões antigas do archC.

Para testar este simulador utilizamos o avr-gcc, ferramenta GNU baseada no GCC que compila código fonte na linguagem C para instruções de avr. Mas infelizmente esta ferramenta não gerava um binário de forma legível, com a qual o archC pudesse trabalhar.

Então passamos a utilizar apenas o montador de instruções o tavasm. Esta ferramenta também está sob licença GLP e é baseada no ARV-Studio, IDE oficial da Atmega que roda sobre o sistema operacional Windows®.

Esta ferramenta gera um binário archC-compatível, porém quando tentamos gerar um código hexadecimal para que pudéssemos examina-lo, percebemos que esta ferramenta gerava endereços de instruções inválidos. Ou seja, ela não levava em conta que o tamanho de instrução do Atmega é de 16bits e sim realiza a contagem de endereços entre instruções como palavras de 8 bits. Ou seja, as instruções viam separadas por um, quando deveriam distanciar-se por dois endereços de wmemória. Este problema foi facilmente contornado utilizando as ferramentas "bc"e "sed", presentes por padrão em qualquer distribuição linux.

É importante ressaltar que o script criado ficará disponível sob a licença GPL, sendo possível a sua utilização, modificação e redistribuição. Segue o script abaixo:

```
#!/bin/bash
arq=$1
tmp=‘date | md5sum | cut -c0-32‘
fname="/tmp/$tmp"
################################
# Puting 40 nop’s in file
################################
for i in $(seq 44); do
echo nop >> $fname
done

cat $arq >> $fname
mv $fname $arq
tavrasm -g $arq

arq=‘echo $arq | sed ’s/\.as$/.gen/’‘
```

```
##################################
# Multiplicate PC's by 2
##################################
while read LINHA
do
num=‘echo $LINHA | cut -c0-6 | tr [a-z] [A-Z]‘
com=‘echo $LINHA | cut -c8-20‘
numH=‘echo -e "ibase=16; $num * 2" | bc‘

numH2=‘printf "%08x" $numH‘

echo "$numH2 $com" >> $fname

done < $arq
mv $fname $arq
```

Também utilizamos arquivos de teste que foram gerados durante a dissertação de mestrado da Andreia Barbiero. Estes arquivos foram disponibilizados pelo Dalton Project e constituem-se de programas clássicos de ordenação, um gerador de números primos, todos descritos na Tabela X. É importante ressaltar que estes mesmos programas ja foram tilizados pela equipe de desenvolvedores do ArchC em testes com modelos de microprocessadores como por exemplo o Intel 8051.

Programa	Descrição do Programa
negcnt.c	Conta 100 númros negativos
int2bin.c	Traduz um número de 16 bits (hexa) para binário
gcd.c	Comparada, soma e subtrai 2 números
cast.c	Trasnforma long em char
fib.c	Gera 100 elementos da série de fibonacci
bubble.c	Ordena 1000 números pelo Bubble sort
insertion.c	Ordena 1000 números pelo Insertion Sort
merge.c	Ordena 1000 números pelo Merge Sort
quick.c	Ordena 1000 números pelo Quick Sort
selection.c	Ordena 1000 números pelo Selection Sort
shell.c	Ordena 1000 números pelo Shell Sort
crivo.c	Gera números primos menor que 1000 pelo Crivo de Eratóstenes

Tabela 6.1

Um outro problema encontrado na migração da versão 1.6 para a versão 2.0 foi contornado ao corrigir o tamanho disponibilizado para a memória de dados. Na versão disponibilizada para os autores deste trabalho, a memória de dados iniciava em 0xA000 e ia até 0xEFFF. Se pararmos para analisar, isso corresponde a mais do que os 8192 bytes necessários. Este problema fazia com que, nesta versão do archC, a memória de dados fosse sobreescrita pela memória de programa e com isso três dos arquivos de testes executavam de maneira incorreta (a_insertion_at.hex, b_merge_at.hex, c_quick_at.hex) sendo que o primeiro deles entrava em loop enquanto os outros dois acessavam posições inválidas da memória ocasionando falha de segmentação. Para identificarmos tal fato foi necessário analisar logs com tamanhos médios de 1GB até 10GB pois os programas de testes que não funcionavam executacam mais de 1 bilhão de instruções. Após verificarmos tal problema, mudamos a memória de dados para ser iniciada em 0xD000 e ir até 0xEFFF, possuindo então os 8192 bytes necessários (Os valores F000 até FFFF são utilizados como pilha pelo stack pointer). Tal mudança solucionou o problema permitindo que os três programas de testes fossem executados corretamente. A seguir descrevemos o código alterado e qual a sua relevância no projeto do simulador:

```
#define OFFSET 0xD000 //Inicio da memoria de dados
```

```
//Le uma word(16 bits) da memória adicionando o OFFSET para acessar a area de dados
int DMread(int index, ac_memport<short unsigned int,unsigned char> *DM)
{
    int data = 0;
    if (index == 0xf014)
        getchar();
    //Compatibiliza o endereço
    index += OFFSET;
    index &= 0xffff;
    data = DM->read(index);
    printf("READ-----------DATA:%x---INDEX:%x\n",data,index);
    return data;
}

//Escreve uma word(16 bits) na memória adicionando o OFFSET para acessar a area de dados
void DMwrite(int index,int data, ac_memport<short unsigned int,unsigned char> *DM)
{
   //Compatibiliza o endereço
   index += OFFSET;
   index &= 0xffff;
   DM->write(index,data);
   printf("READ----------OOOOIIII: %d",DM->read(index));
   printf("WRITE-----------DM:%x---INDEX:%x\n",data,index);
}

//Le um byte da memória adicionando o OFFSET para acessar a area de dados
int DMread_byte(int index, ac_memport<short unsigned int,unsigned char> *DM)
{
    int data = 0;

    index += OFFSET;
    index &= 0xffff;
```

```
    data = DM->read_byte(index);
    printf("READ BYTE-----------DATA:%x---INDEX:%x\n",data,index);
    return data;
}

//Escreve um byte na memória adicionando o OFFSET para acessar a area de dados
void DMwrite_byte(int index,int data, ac_memport<short unsigned int,unsigned char> *DM)
{
   //Compatibiliza o endereço
   index += OFFSET;
   index &= 0xffff;
   DM->write_byte(index,data);
   printf("WRITE BYTE-----------DM:%x---INDEX:%x\n",data,index);
}
```

Este trecho de código é o único que utiliza o OFFSET alterado, que inicialmente apresentava o valor A000, sendo corrigido para D000. Tanto para escrever na memória ou para ler da mesma o OFFSET é utilizado, pois calcula o endereço para acessar a área de dados.

Houve ainda outro problema quando manipulando endereços da pilha com a instrução rcall e rjmp descrita abaixo:

```
void ac_behavior( rcall )
{

        int addr = 0;



        printf("@@@@@@@@%s\n",get_name());
        printf("STACK: 0x%x\t\tantes\n",DMread(SP.read(),&DM));
        printf("SP = 0x%x\t\tantes\n", SP.read());
        printf("K12 = 0x%x\t\tantes\n", k_12);
        DMwrite(SP.read(), pc,&DM);
```

```
        if (k_12 >> 11)
        {
                addr = k_12 | 0xF000;
                pc -= 2;
        }
        else
                addr = k_12;
        pc += addr * 2;

        pc &= 0xffff;
        ac_pc = pc;

        printf("STACK: 0x%x\t\tdepois\n",DMread(SP.read(),&DM));
        SP -= 2;
        printf("SP = 0x%x\t\tdepois\n", SP.read());

        //Passa para a próxima instrução
        NEXT = 1;
}
```

Podemos ver que a instrução rcall manipula endereços do program counter, atribuindo um novo valor ao pc de acordo com o valor passado a instrução rcall. Este valor é constituído de 12 bits e atribuído à variável k_12. Porém devemos lembrar que um rcall pode conter endereços relativos maiores ou menores do que o endereço atual e portanto é necessário tratar o overflow, pois o mesmo pode ocorrer ao somar um valor de k_12 que ultrapasse FFFF. Na versão recebida pelos autores deste trabalho, o código estava da seguinte maneira:

```
void ac_behavior( rcall )
{

...(código igual)
```

```
        addr = k_12;
        pc += addr * 2;
        pc &= 0xffff;
        ac_pc = pc;

...(código igual)

}
```

Ou seja, não era feito nenhuma verificação (a qual está sendo feita agora no IF/ELSE com a ajuda do SHIFT da linguagem C) para manipular o overflow. Na versão recebida como podemos ver apenas multiplicava-se o endereço relativo por 2 (tendo em vista que o program counter cresce de 2 em 2 e por isso não existem endereços ímpares) e então o endereço multiplicado era atribuído ao pc.

Para descobrir tal problema também foi necessário avaliar arquivos de log com tamanhos na casa dos gigabytes, pois os problemas só ocorreram com arquivos de teste que executavam mais de 1 milhão de instruções, que utilizavam o rcall e ao fazerem isto manipulavam endereços relativos menores do que o endereço atual (necessitando diminuir o endereço contído no program counter e não aumentar). Os arquivos que apresentaram problema foram: b_merge_at.hex e c_quick_at.hex.

6.4 MOTOROLA DSP56F827

O trabalho em cima desta arquitetura foi o que despendeu maior tempo e esforço por parte dos autores deste documento. Isto deve-se ao tamanho do conjunto de instruções da arquitetura (aproximadamente 350 instruções) e da complexidade das operações que cada uma destas instruções realiza. Um dos pontos importantes foi detectar um seg fault apresentado no momento da geração do simulador na versão 2.0.

Esta falha de segmentação acontecia devido a uma descrição de tipo (ou grupo) de instrução que possuia muitas intruções, ultrapassando o tamanho máximo por linha na leitura do arquivo, mas que nas versões anteriores, como a 1.6 na qual o modelo foi inicialmente desenvolvido, era possível fazer descrições de tamanho ilimitado.

Além de realizarmos o processo de migração da arquitetura para um modelo compatível com

a arquitetura 2.0 do archC, desenvolvemos também (através de ferramentas disponíveis pela própria equipe do archC) um montador de código objeto baseado no "GNU gas", processo o qual será descrito no próximo capítulo.

Um montador, utiliza um processo montagem que recebe como entrada um arquivo texto com o código fonte do programa em assembly e gera como saída um arquivo binário, o módulo objeto, contendo o código de máquina e outras informações relevantes para a execução do código gerado. Em geral, montadores oferecem facilidades além da simples tradução de código assembly para código de máquina. Além das instruções do processador, um programa fonte para o montado pode conter diretivas ou pseudo-instruções definidas para o montador (e não para o processador), assim como macro-instruções, uma seqüência de instruções que será inserida no código ao ser referenciada pelo nome. Um montador que suporte a definição e utilização de macro-instruções é usualmente denominado um macro-montador (macro-assembler). Um montador multiplataforma (cross-assembler) é um montador que permite gerar código para um processador-alvo diferente daquele no qual o montador está sendo executado. Na seqüência apresenta-se brevemente as atividades relacionadas ao processo de montagem, partingo da descrição do formato de entrada esperado até a geração do módulo-objeto de saída.

Capítulo 7

Processo de geração dos simuladores

Para gerar um simulador de uma arquitetura qualquer com o archC são necessários 2 passos:

- Geração do simulador
- Compilação

7.1 Gerando o simulador

No primeiro passo é necessário indicar qual o tipo de simulador que será gerado. Para isso são disponibilizados as ferramentas acsim e actim. A principal diferença entre ambas é o controle de ciclos, portanto o acsim é utilizado para gerar simuladores monociclos enquanto o actsim é utilizado para gerar simuladores multiciclos ou segmentados. Além disso outras opções estão disponíveis como -abi, -dasm, -g, -dd, -h, -ndc, -s, -vb, -vrs e -gdb. Destaque para as opções -abi - A qual indica que uma interface para chamadas de sistemas esta ativada, -g - A qual ativa o modo debug (traces e logs), -gdb - A qual ativa o modo debug para ser utilziado durante a simulação e -s - A qual ativa o modo de coleção de estatísticas durante a simulação.

Neste trabalho os arquivos para geração dos simuladores foram gerados com o seguinte comando:

```
acsim [modelo].ac
```

Através deste comando, o archC interpreta o arquivo de descrição da arquitetura, gerando todos os arquivos necessários para a construção do simulador, excetuando o arquivo que descreve as ações de cada instrução. É importante ressaltar que até este passo os 3 modelos (Atmel,

Rabbit e Motorola) obtiveram sucesso, sendo inclusive gerados o Makefile e os arquivos de configuração. Para tanto foram corrigidas mais de 1000 linhas de erros, decorrentes das mudanças entre a versão archC 1.6 e 2.0.

7.2 Compilação

Após a geração dos sources do simulador através da ferramenta acsim, é necessário compilar-lo. Para isto, utiliza-se o seguinte comando:

```
make -f Makefile.archc
```

Para este comando apenas o modelo Atmel foi compilado com sucesso pois foram necessários inúmeros testes e comparações adequadas entre resultados esperados dos arquivos de testes. Além disso parte o tempo que seria utilizado na completa migração da versão 1.6 para a versão 2.0 do Rabbit e do Motorola foi utilizado na geração do montador para o Motorola que será descrita no capítulo a seguir.

Capítulo 8

Geração do montador para DSP56F827

8.1 Motivação.

No caso da geração do montador, existe um importante detalhe não documentado no site do archC onde apenas "novos formatos"são aceitos no script asmgen.sh. Um dos principais problemas descritos foi identificar tal formato. Além disso, tivemos alguns bons desafios durante o processo de geração do gas. Um destes problemas foi que devido à inconsistências nos headers do arquivo de descrição das instruções, tivemos que "desmembrar"todas as instruções, colocando-as em seu formato binário, afim de identificar a quantidade e os valores dos bits utilizados em cada campo das instruções compiladas. Após isto comparamos os bits com os valores reais encontrados em executáveis e indetificamos as discrepâncias. Estas instruções foram corrigidas afim de garantir um montador confiável.

Outro problema para gerar o montador foi que não foi possível localizar a documentação sobre o arquivo de descrição das instruções da arquitetura. Neste caso fomos obrigados a proceder de forma não convencional, realizando vários testes com padrões que julgávamos dotados de sentido, afim de identificar o padrão de descrição correto. Este padrão será comentado a seguir neste documento.

8.2 Correta descrição dos padrões da arquitetura para a geração automática do Gnu-Assembler.

8.2.1 Descrição do mapa de registradores

Primeiramente devemos definir corretamente o mapa de registradores, e identificar qual será o endereço de cada um deles. Por exemplo, precisamos dizer que o registrador *A0* é, na verdade, o registrador 1 do banco de registradores. Para isto usaremos a instrução "ac_asm_map reg"do arquivo "dsp56827_isa.ac". O resultado será algo como:

```
ac_asm_map reg {
      "X0" = 0;
      "Y0" = 2;
      "Y1" = 6;
      "A"[0..2] = [4..6];
      "B"[0..2] = [9..11];
      "A" = 4;
      "B" = 5;
      "R"[0..31] = [0..31];
      "FP" = 30;
      "SP" = 27;
}
```

Dizemos aqui que o registrador X0 será mapeado para o registrador número 0, o Y0 para o registrador número 2, e assim sucessivamente.

O nome ***reg*** aqui define o nome da classe registradores. Esta informação será útil mais tarde quando formos definir as instruções propriamente ditas. Munidos disto, se necessário, podemos dizer que certo parâmetro de certa instrução poderá receber apenas registradores do tipo ***%reg***, e outro dizendo que poderá receber apenas registradores do tipo ***%reg_acumulacao***.

8.2.2 Descrição das classes de instruções

Desde as versões mais antigas do ArchC é possível definir grupos de instruções, afim de facilitar o processo de descrição binária dos campos. Ou seja, instruções de formato semelhante (mesmo número de bits para cada campo), podem ser agrupadas em um tipo, para que não seja necessário

definir várias vezes os mesmos campos.

Estes grupos devem ser descritos no arquivo **"dsp56827_isa.ac"**, na estrutura inicial do arquivo como no exemplo abaixo.

```
AC_ISA(dsp56827){
  //Ex:add A,B
  ac_format Type_1W  = "%op4:16";
  ac_format Type_2W  = "%op4:16 %imm16:16";
}
```

Vemos acima dois formatos de instruções, o **Type_1W** e o **Type_2W**.

Percebemos que estes formatos de instruções possuem alguns parâmetros, que são definidos por:

```
%operator:numero_de_bits
```

Onde *operador* é o nome do operador (ou do parâmtro); **:** é o separador, e *numero_de_bits* é o tamanho em bits do campo. Utilizamos o % para identificar o início do nome de cara parâmtro.

Ou seja, as instruções do tipo *Type_2W* recebem dois parâmetros, um chamado op4 e outro chamado imm16, ambos com 16 bits de tamanho.

Não importa, aqui, o tipo do campo, o tamanho ou o valor que eles recebem. Se definirmos um operador de instrução com tamanho de 2 bits, e enviarmos para ele um valor de 32bits, este valor será truncado no lado mais significativo. Ou seja, neste momento o archC interpretará os numeros binários **"0000010"** e **"010"** como **"10"** para operandos de tamanho 2.

8.2.3 Descrição das instruções

Com os conhecimentos adquiridos até aqui podemos então definir uma instrução. Tomaremos como exemplo a instrução ***"move"***, mais especificamente a instrução "move"definida pelo formato "*movep_reg*"no arquivo de descrição da arquitetura.

As intruções deverão ser escritas dentro do bloco ***ISA_CTOR(dsp56827)***, que por sua vez estará dentro do bloco ***AC_ISA(dsp56827)*** dentro do arquivo **"dsp56827_isa.ac"**.

```
ISA_CTOR(dsp56827){
movep_reg.set_asm("move p:(%reg)%reg,%reg",rr,m,hhhh);
```

```
movep_reg.set_decoder(op1=0xe,W=0x2);
```

Pelo exemplo acima, notamos que as instruções são definidas utilizando os métodos set_asm e set_decoder. Descrevemos estes métodos em breve.

Sabemos também que esta instrução possui o formato ***Type_Movep*** descrito abaixo:

```
ac_format Type_Movep = "%op1:4 %hhhh:4 %W:4 %m:2 %rr:2";
```

Esta informação será bastante útil para explicar o exemplo.

8.2.3.1 O método set_ast

```
movep_reg.set_asm("move p:(%reg)%reg,%reg",rr,m,hhhh);
```

Podemos ver claramente neste exemplo que esta instrução recebe como parâmetro três registradores (definidos pela "ac_asm_map") da classe reg.

Ou seja, a instrução poderá ser chamada na forma:

```
move p:(A0)A1,A3
```

Pois A0, A1 e A3 são registradores definidos anteriormente no mapa de registradores pela a classe ***reg***. Ou seja, poderemos ter aqui qualquer um dos registradores desta classe.

Neste momento, o archC realiza uma análise semântica dos elementos, e preeche os valores de *rr,m* e *hhhh* com 1,2,3 respectivamentes, pois estes são os valores referentes à posição dos registradores utilizados no exemplo no banco de registradores.

8.2.3.2 O métodos set_decorder

Percebemos que os parâmtros *rr,m* e *hhhh* foram preenchidos, mas o formato de instrução ***Type_Movep*** exige 5 parâmetros. Os outros parâmetros definidos na instrução, tais como op1 e W, deverão ser preenchidos "forçadamente"através do método set_decoder. No caso, estes campos serão preenchidos pelos valores 0xE e 0x2. Se por ventura algum campo for definido mas não foi preenchido, este será automaticamente preenchido pelo valor 0 no assembler.

8.2.4 Conclusão

Munidos da informação desta seção, podemos agora descrever a arquitetura completa de todas as instruções.

Uma descrição mais completa da arquitetura pode ser encontrada, em caráter de exemplificação, no Anexo 12.1

8.3 Gerando o montador

8.3.1 Considerações iniciais

Primeiramente, geraremos o montador à partir do código fonte do GAS (GNU Assembler), logo é necessário obter o código fonte do mesmo como descrito na seção 4.3.

Em caráter de exemplificação, supomos que o diretório com os fontes do binutils estejam em um diretório de nome *binutils*, juntamente com o arquivo de descrição da arquitetura, no caso, o **dsp56827.ac**.

8.3.2 Alterando binutils com asmgen.sg

Para gerar o montador, utilizaremos uma ferramenta específica do archC, chamada ***asmgen.sh***. Esta ferramenta deverá ser invocada da forma:

```
$archC/archc/bin/asmgen.sh dsp56827.ac mymoto
```

A variável $archC deverá contar o caminho para a árvore de diretórios do archC

O que esta ferramenta irá fazer é alterar o código fonte do binutils de forma à acrescentar uma meta-arquitetura (chamada, no caso, de mymoto) descrita no arquivo ***dsp56827.ac***. Agora o binutils está apto à gerar montadores para qualquer arquitetura possíveis de ser descritas sob à linguagem do archC.

8.3.3 Compilando o bintuils para a nova arquitetura

Para compilar corretamente o binutils para a nova arquitetura, serão necessários dois diretórios:

- build-dir
- install-dir

Como os nomes já dizem, estes diretórios servem para construir e compilar a árvore do binutils, e gravar o arquivo binário, respectivamente.

Logo deve-se criar os dois diretórios e, dentro do diretório build-dir, executar o comando:

```
$archC/binutils-2.16/configure --target=mymoto --prefix=$PWD/../install-dir
```

Após isto, executa-se a sequência de comandos padrão para compilação do binutils:

```
make all-gas
make install-gas
```

8.3.4 Script para automatizar o processo

Para automatizar o processo de geração do montador, foi criado um script que leva em conta as modificações do arquivo ***dsp56827.ac***, gerando a cada execução um montador na pasta *install-gas*

```
#!/bin/bash
rm -rf build-dir install-dir binutils
$arch/archC/bin/asmgen.sh dsp56827.ac mymoto
mkdir build-dir
mkdir install-dir
cd build-dir
$archC/binutils-2.16/configure --target=mymoto-elf --prefix=$PWD/../install-dir
make all-gas
make install-gas
```

É necessário remover os diretórios build-dir, install-dir e binutils se deseja-se gerar o montador com o mesmo nome. Se um montador com novo nome for gerado, a segunda linha poderá ser desconsiderada.

Capítulo 9

RESULTADOS OBTIDOS

Neste capítulo descreveremos os resultados obtidos com o modelo funcional do Atmega na versão 2.0 Beta do ARCHC. É importante lembrar que a comparação de resultados se dará em relação à um modelo multiciclo e ainda na versão 1.6 do ARCHC. É importante lembrar que todos os arquivos de testes foram executados com sucesso e consequentemente obtiveram o mesmo número de instruções em ambas as versões do Atmega.

9.1 ATMEL MULTICICLO COM ARCHC 1.6 X ATMEL FUNCIONAL COM ARCHC 2.0

Programa	**Nº Ciclos 1.6**	**Nº Instruções 1.6**	**Nº Ciclos 2.0**	**Nº Instruções 2.0**
negcnt.c	24064	14060	14060	14060
int2bin.c	1055	1055	1055	1055
gcd.c	199784	119041	119041	119041
cast.c	287	199	199	199
fib.c	13121	8230	8230	8230
bubble.c	54,8*1000000	35279030	32279030	35279030
insertion.c	21,3*1000000	16656296	16656296	16656296
quick.c	1*1000000	625292	625292	625292
selection.c	30*1000000	19246041	19246041	19246041
shell.c	9,46*1000000	6149869	6149869	6149869
crivo.c	20,3*1000000	6149869	6149869	6149869

Capítulo 10

Conclusão

Durante o longo período de desenvolvimento com as ferramentas disponíveis através do archC, concluímos que este é uma opção viável para desenvolvimento de simuladores para arquiteturas específicas em dispositivos embarcados.

Através do asmgen.sh, é possível criar rapidamente um montador para qualquer arquitetura descritível pela ferramenta do archC.

Capítulo 11

Bibliografia

11.1 Sítios

11.1.1 ArchC

http://www.archc.org

Website do archC

11.1.2 SystemC

http://www.systemC.org

Website do systemC

11.1.3 GNU Binutils

http://www.gnu.org/binutils

Informações e download do código fonte do conjunto de ferramentas do GNU-Binutils

11.1.4 Tavrasm

http://www.tavrasm.org

Montador opensource para a família de microcontroles AVR-Atmel.

11.1.5 Dalton Project

http://www.cs.ucr.edu/ dalton/

O Dalton project foca-se no design de propriedade intelectual baseados em sistemas computacionais embarcados.

11.2 Publicações

Andréia A Barbiero e R A Hexsel. Ambiente de Suporte ao Projeto de Sistemas Embarcados.

VII Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD 2006), 18-20out 2006.

Andréia A Barbiero e R A Hexsel. Ambiente de Suporte ao Projeto de Sistemas Embarcados.

Dissertação de Mestrado, jul06;

Capítulo 12

Documentos Anexos

12.1 Exemplo de dsp56827_isa.ac

```
/*****************************************************/
/* The ArchC SPARC-V8 functional model.              */
/* Author: Sandro Rigo and Marcus Bartholomeu        */
/*                                                   */
/* This file is automatically generated by ArchC     */
/* WITHOUT WARRANTY OF ANY KIND, either express      */
/* or implied.                                       */
/* For more information on ArchC, please visit:      */
/* http://www.archc.org                              */
/*                                                   */
/* The ArchC Team                                    */
/* Computer Systems Laboratory (LSC)                 */
/* IC-UNICAMP                                        */
/* http://www.lsc.ic.unicamp.br                      */
/*****************************************************/


AC_ISA(dsp56827){

  //Ex:add A,B
  ac_format Type_1W  = "%op4:16";
```

```
//Ex: inc X:0000
ac_format Type_2W  = "%op4:16 %imm16:16";
//Ex: bfclr #0001,X0
ac_format Type_2W_Bit1  = "%op2:8 %mod:3 reg:5 %imm16:16";
//Ex: brclr #01,X0,#11
ac_format Type_2W_BitBr1  = "%op2:8 %mod:3 reg:5 %mask8w2:8 %addr7w2:8";
//Ex: bfclr #0001,X:(R2+pp6) ou bfclr #0001,X(SP-aa6)
ac_format Type_2W_Bit2  = "%op2:8 %dif:2 %pp6:6 %imm16:16";
//Ex: brclr #01,X:(R2+pp6),#11 ou brclr #01,X(SP-aa6),#11
ac_format Type_2W_BitBr2  = "%op2:8 %dif:2 %pp6:6 %mask8w2:8 %addr7w2:8";
ac_format Type_2W_Bit3  = "%op2:8 %dif2:1 %pp7:7 %imm16:16";
ac_format Type_2W_BitBr3  = "%op2:8 %dif2:1 %pp7:7 %mask8w2:8 %addr7w2:8";
ac_format Type_Dalu_pp6_SP = "%op1:4 %dc2:2 %fff:3 %dc9:1 %pp6:6";
ac_format Type_Dalu_fff_pp6_SP = "%op1:4 %dc2:2 %fff:3 %dc9:1 %pp6:6 %rsv:16";
ac_format Type_Dalu_pp6_SP_fff = "%op1:4 %dc2:2 %fff:3 %dc9:1 %pp6:6";
ac_format Type_Dalu_SP = "%op1:4 %dc2:2 %fff:3 %dc9:1 %pp6:6";
ac_format Type_Dalu_imm5 = "%op1:4 %dc2:2 %fff:3 %dc10:2 %bb:5";
ac_format Type_Dalu_imm16 = "%op1:4 %dc2:2 %fff:3 %dc11:7 %imm16:16";
ac_format Type_Spm = "%op5:2 %w:1 %kk:2 %hhh:3 %F:1 %jjj:3 %x:1 %m2:1 %rsp:2";
ac_format Type_Dalu3op2 = "%op2:8 %f0:1 %qqq:3 %mod2:2 %f1f2:2";
ac_format Type_Movep = "%op1:4 %hhhh:4 %W:4 %m:2 %rr:2";

ac_instr<Type_1W> add_b_a, add_a_b, tfr_b_a, tfr_a_b, sub_y_a;
ac_instr<Type_1W> rol_y1, asr_x0, asr_y0, asr_y1, ror_x0, lsr_x0, lsr_y0, lsr_y1;
ac_instr<Type_1W> rts, rti, nop, debug, stop1, wait1, swi, illegal;
ac_instr<Type_1W> enddo, norm_a, norm_b;
ac_instr<Type_2W_BitBr3> brclr_imm_pp7, brset_imm_pp7;
ac_instr<Type_3W_BitBr> brclr_imm_imm, brset_imm_imm;
ac_instr<Type_Movep> movep_reg,move_reg_p;
ac_instr<Type_Move_Rn_N> movex_rsp_N,mover_rsp_N;
ac_instr<Type_Move_Rp_mm> movex_rp_mm,mover_rp_mm;
ac_instr<Type_Move_reg_reg> move_reg_reg;
```

```
ac_instr<Type_Dalu3op> mpy_dalu3op,mac_dalu3op,mpyr_dalu3op,macr_dalu3op;


// registradores
ac_asm_map reg {
    "X0" = 0;
    "Y0" = 2;
    "Y1" = 6;
    "A"[0..2] = [4..6];
    "B"[0..2] = [9..11];
    "A" = 4;
    "B" = 5;
    "R"[0..31] = [0..31];
    "FP" = 30;
    "SP" = 27;
}


ISA_CTOR(dsp56827){

  movep_reg.set_asm("move p:(%reg)%reg,%reg",rr,m,hhhh);
  movep_reg.set_decoder(op1=0xe,W=0x2);

  movex_imm.set_asm("move x:%imm,%reg",imm16,d5cap);
  movex_imm.set_decoder(op1=0xf,dc4=0x5);

  mover_imm.set_asm("move %reg,x:%imm",d5cap,imm16);
  mover_imm.set_decoder(op1=0xd,dc4=0x5);

  move_reg_reg.set_asm("move %reg,%reg",d5orig,d5cap);
  move_reg_reg.set_decoder(op1=0x8,dc6=0x0);
```

```
movex_pp7.set_asm("move x:<<%reg,%reg",pp7,hhhh);
movex_pp7.set_decoder(op1=0xb,dif2=0x0);

mover_pp7.set_asm("move %reg,x:<<%reg", hhhh, pp7);
mover_pp7.set_decoder(op1=0x9,dif2=0x0);

movex_r2.set_asm("move x:(r2+<<%reg),%reg", pp6, hhhh);
movex_r2.set_decoder(op1=0xb,dif=0x2);

mover_r2.set_asm("move %reg,x:(r2+<<%reg)", hhhh, pp6);
mover_r2.set_decoder(op1=0x9,dif=0x2);

movex_sp.set_asm("move x:(sp-<<%reg),%reg", pp6, hhhh);
movex_sp.set_decoder(op1=0xb,dif=0x3);

mover_sp.set_asm("move %reg,x:(sp-<<%reg)",hhhh, pp6);
mover_sp.set_decoder(op1=0x9,dif=0x3);

move_imm7.set_asm("move %reg,%reg", pp7, hhhh);
move_imm7.set_decoder(op1=0xc,dif2=0x0);

add_imm5.set_asm("add <<%imm,%reg", bb, fff);
add_imm5.set_decoder(op1=0x4,dc2=0x1,dc10=0x0);

add_imm16.set_asm("add %imm,%reg", imm16, fff);

bfclr_imm_pp7.set_asm("bfclr %imm,X:<<%reg", imm16, pp7);
bfclr_imm_pp7.set_decoder(op2=0xa1,dif2=0x1);

bfset_sp_aa6.set_asm("bfset %imm,X:(SP-<<%reg)", imm16, pp6);
bfset_sp_aa6.set_decoder(op2=0xa2,dif=0x3);
```

```
brclr_r2_pp6.set_asm("brclr %imm,X:(R2+<<%reg),%imm", mask8w2, pp6, addr7w2);
brclr_r2_pp6.set_decoder(op2=0xaa,dif=0x2);

brclr_mask8_d5.set_asm("brclr %imm,%reg,%imm", mask8w2, reg, addr7w2);
brclr_mask8_d5.set_decoder(op2=0x8b,mod=0x6);

brclr_imm_imm.set_asm("brclr %imm,X:imm16w3,%imm", mask8w3, addr7w3);
brclr_imm_imm.set_decoder(op4=0x8af4);

brclr_imm_pp7.set_asm("brclr %imm,X:<<%reg,%imm", mask8w2, pp7, addr7w2);
brclr_imm_pp7.set_decoder(op2=0xab,dif2=0x1);

brset_sp_aa6.set_asm("brset %imm,X:(SP-<<%reg),%imm", mask8w2, pp6, addr7w2);
brset_sp_aa6.set_decoder(op2=0xae,dif=0x3);

cmp_a1_y0.set_asm("cmp A1,Y0");
cmp_a1_y0.set_decoder(op4=0x7c23);

clr_a.set_asm("clr A");
clr_a.set_decoder(op4=0x6c30);

clr_b.set_asm("clr B");
clr_b.set_decoder(op4=0x6cb0);

asl_a.set_asm("asl A");
asl_a.set_decoder(op4=0x7430);

asl_b.set_asm("asl B");
asl_b.set_decoder(op4=0x74b0);

rol_a.set_asm("rol A");
rol_a.set_decoder(op4=0x7630);
```

```
    do_reg.set_asm("do %reg,%reg", d5, addr16);
    do_reg.set_decoder(op2=0xcc,mod=0x6);

    jmp.set_asm("jmp %reg", addr16);
    jmp.set_decoder(op1=0xe,cccc=0x9,dif=0x2,dc=0x1);

    jcc.set_asm("j$cccc %reg", addr16);
    jcc.set_decoder(op1=0xe,dif=0x2,dc=0x1);

    jsr.set_asm("jsr %reg", addr16);
    jsr.set_decoder(op2=0xe9,dif=0x3,dc=0x2);
  };
};
```