Escola Politécnica da Universidade de São Paulo
Departamento de Engenharia de Sistemas Eletrônicos - PSI

PSI-2553- Projeto de Sistemas Integrados

Experiência 2: O Processador Plasma (Parte Prática)

M.S. / W.J.C / M.A.R.J(17)

Conteúdo:

1. Objetivos

Esta experiência visa a familiarização do estudante com a arquitetura do processador Plasma, que segue a arquitetura MIPS, introduzida nas aulas de teoria. Serão realizados o estudo de programa compilado (*assembly*) e a simulação de uma implementação específica do processador que adota a técnica de segmentação (*pipelining*). Após a experiência, o aluno saberá correlacionar o código em *assembly* e o fluxo de dados e controle dentro do processador.

2. Parte Experimental

2.1 Aprontando os Arquivos de Compilação, Compiladores e Ferramentas do Plasma

Esta parte é semelhante àquela feita no tutorial do Plasma.

1) Criar o diretório **X:\psi2553\exp2\infra.**
2) Copiar para **X:\psi2553\exp2\infra** o arquivo **mips_tools.zip** que está em **Rede\NEWSERVERLAB\psi2553\exp2**
3) Na subdiretório **X:\psi2553\exp2\infra** extraia as pastas comprimidas que estão em **mips_tools.zip**. Com isso duas pastas deverão aparecer: **X:\psi2553\exp2\infra\tools** e **X:\psi2553\exp2\infra\gccmips_elf**. Atenção: toda compilação e montagem será feita no subdiretório **X:\psi2553\exp2\infra\tools**, porém teremos que usar os programas via linhas de comando (diretamente no DOS).

2.2 Conhecendo os arquivos a serem compilados

Os arquivos copiados estão na forma original da distribuição do Plasma encontrados no link www.opencores.com. São um tanto diferentes dos vistos no tutorial do Plasma, usado em aulas anteriores, uma vez que estes últimos são arquivos "limpos", onde uma série de detalhes foi removida com o intuito de facilitar o primeiro contato e familiarização com o processador.

Atenção: quando há uso de sistema operacional, o código correspondente é instalado na posição 0x0 da memória. Como não estamos trabalhando com sistema operacional, a posição 0x0 é ocupada pelo código de **boot.asm**. Depois os arquivos de apoio, **no_os.c** e **plasma.h** junto com o arquivo de programa do usuário (**fibonacci.c**, nesta experiência) garantem o funcionamento de todo o software do Plasma.

1) Em **X:\psi2553\exp2\infra\tools**, examinar o arquivo **boot.asm**. Este arquivo em *assembly* contém algumas instruções colocadas na parte inicial da memória. Observe e tenha certeza que entende o funcionamento dos trechos definidos pelos seguintes labels:
   a. $BSS_CLEAR
   b. $L1

→ No relatório:

DADOS EXPERIMENTAIS:

Impressão do arquivo boot.asm somente com os trechos de a até b acima.

ANÁLISE E DISCUSSÃO:

a) **Identifique os trechos acima no código impresso.**

b) **Explique o objetivo do código do trecho a.**

c) **Explique o objetivo do código do trecho b.**

2) Em **X:\psi2553\exp2\infra\tools**, examinar rapidamente arquivo **no_os.c**. Este arquivo em C contém algumas rotinas de uso geral e de interrupção. Voltaremos a examiná-lo com mais detalhes em outra oportunidade.

3) Como ocorrido no uso do tutorial, o arquivo **makefile** já está ajustado para gerar o código do programa de cálculo do número de Fibonacci. Examine o **makefile** em **X:\psi2553\exp2\infra\tools**,

4) Na mesma pasta observar e editar o código do cálculo de Fibonacci em **fibonacci.c**. Observe que há o programa *main* e a função fibo chamada dentro da primeira.

5) Vamos realizar modificações no código de **fibonacci.c** para inserir algumas instruções de leitura à memória. Faça o seguinte procedimento:

- Calcule num_aluno= No_USP mod 21

  * Caso o num_aluno obtido seja igual a 0 => use num_aluno= 21

  a. Introduza no código C do *main* o seu num_aluno;

  b. Dentro da sub-rotina *fibo* apague parte das instruções x[k] = sequência de números primos, de forma a manter apenas a sequência de 1 a num_aluno. Por exemplo, se o seu num_aluno=3, você deve manter apenas as linhas correspondentes a x[1], x[2] e x[3]. As outras devem ser apagadas ou comentadas.

→ No relatório:

DADOS EXPERIMENTAIS:

Impressão do arquivo fibo.c com a mudança realizada.

ANÁLISE E DISCUSSÃO:

a) **Mostre o cálculo de num_aluno;**

b) **Identifique as inserções realizadas com o número 'num_aluno' acima no código impresso.**

2.3 Pós-compilação e interpretação do código *assembly* do programa fibonacci

1) Seguir os passos apresentados no tutorial do Plasma para realizar a compilação do código todo através do arquivo **makefile**.

2) Olhar o arquivo **X:\psi2553\exp2\infra\tools\test.map.** Localize os endereços de início da rotina main e da fibo.

→ No relatório:

DADOS EXPERIMENTAIS:

Impressão do arquivo test.map.

ANÁLISE E DISCUSSÃO:

Identifique na impressão os endereços de início da rotina main e da fibo.

3) Olhar o arquivo **X:\psi2553\exp2\infra\tools\test.lst**. Observe o código *assembly*. Localize os seguintes pontos de interesse:
   a. A partir do início do código (posição 0x0), encontre a chamada para o main;
   b. Início do main;
   c. Retorno do main (retorno- descobrir a última instrução a ser executada)
   d. Chamada da função fibo (dento de main)
   e. Início
   f. Fim da função fibo (retorno- descobrir a última instrução a ser executada)
   g. Dentro de fibo, as escritas do vetor x (números primos) na memória
   h. Uso do stack pointer para armazenamento de valores de registradores
   i. Loop final do programa após execução do main.

→ No relatório:

DADOS EXPERIMENTAIS:

Impressão de trechos do arquivo test.lst com os pontos de interesse

ANÁLISE E DISCUSSÃO:

1) **Identificar na impressão os pontos a até i.**
2) **Explique, com a devida identificação de localização, como o argumento de entrada num_entrada é passado na chamada da função fibo. Identifique o(s) registrador(es) correspondentes(s) envolvido(s).**
3) **Explique, com a devida identificação de localização, como o resultados &data_o é retornado pela função fibo. Identifique o(s) registrador(es) envolvido(s).**

2.4 Simulação no Quartus para associação de eventos/sinais a instruções *assembly*

1) Copiar para **X:\psi2553\exp2\projeto** todos os arquivos que estão em **Rede\NEWSERVERLAB\psi2553\exp2\projeto.** Eles correspondem aos arquivos VHDL do sistema plasma e os arquivos de simulação.
2) Olhe os códigos VHDL, principalmente de plasma.vhd, mlite_CPU.vhd e RAM.vhd. Procure entender as interfaces com o auxílio do diagrama da apostila teórica.
3) Na janela DOS, tecle **gmake tohex**. Os arquivos code_k.hex gerados integrarão as memórias RAM, em formato hex, como descrito na parte teórica, são gerados. A interpretação exata do formato hex é dada no Apêndice.
4) Copie os quatro arquivos code_k.hex para **X:\psi2553\exp2\projeto**.
5) Neste subdiretório, edite o arquivo RAM.vhd. para incluir os quatro arquivos hex. Para isto, encontre a descrição das quatro memórias da Altera e faça a edição, se necessário. Atenção: O arquivo *code_0.hex* corresponde ao conteúdo do byte mais significativo da memória.

→ **No relatório:**

DADOS EXPERIMENTAIS:

Impressão do arquivo pedaço de código VHDL de interesse (do *ram.vhd*)

ANÁLISE E DISCUSSÃO:

Explique como é o mecanismo para a introdução do programa objeto no código VHDL.

6) Carregue o programa *Quartus II* da Altera para fazer as simulações através do seguinte caminho: ***Iniciar/Programas/Altera/Quartus II 9.1.***

7) Crie um projeto como o nome *plasma*. Para isto, use o ***New Project Wizard***. Após atribuir o nome do projeto, incluir todos os arquivos *.vdh, assinalar o dispositivo, na caixa de diálogo seguinte, no campo de simulação, adicione em Tool name: ModelSim-Altera e no Format: VHDL Após finalizar esta etapa poderá observar as características do projeto na janela ***Project Navigator***. Clicando duas vezes sobre o *plasma.vhd* (em ***Files***) poderá visualizar o código vhd do projeto. Lembre-se que os arquivos *.hex já estão incluídos pela referência feita nas memórias RAM.

8) Usaremos o simulador ModelSim nesta experiência pela facilidade de se acessar os sinais internos do projeto. Para configurar o Quartus para este objetivo, acesse o menu ***Assignments > EDAToolSettings > Simulation***.

9) Na opção "Compile test bench":
   a. clique em testbench
   b. clique em NEW
   c. Completar: "Test bench name" = plasma_tbw. "Top level module in test bench" = plasma_tbw. "Design instance name in test bench" = behavior.
   d. No campo "Test bench files", adicionar o arquivo "plasma_tbw.vhd"

10) Na opção "Use script to set up simuation" indique a rota do arquivo "inicio.txt", que será o script para a simulação no ModelSim.

11) Abra os arquivos plasma_tbw.vhd e inicio.txt e os estude cuidadosamente. Use as Figuras 1 e 2 da apostila teórica para associação com os sinais de inicio.txt.

→ **No relatório:**

ANÁLISE E DISCUSSÃO:

Em referência ao arquivo plasma_tbw.vhd, explique o que é a instância u0 e o que faz o processo tb.

12) Compile o arquivo *plasma.vhd*. Selecione no menu ***Processing*** > ***Start > Start Analysis & Synthesis***. O resultado da compilação é apresentado numa janela tipo *pop-up*, pressione ***OK***. Uma janela com o relatório da compilação é aberta automaticamente ou pode ser acessada pela opção do menu ***Processing*** > ***Compilation report***.

13) Realize a simulação funcional do arquivo plasma. Para realizar a simulação funcional: ***Tools > Run EDA Simulation Tool > EDA RTL Simulation***. O ModelSim deverá abrir já com a janela de simulação pronta.

ATENÇÃO: lembrar que estão sendo observados os sinais em cada estágio de um *pipeline*. Portanto existe uma defasagem no tempo entre os sinais que correspondem a

cada instrução. Por exemplo, se o resultado do estágio *fetch* ocorrer num ciclo *i* o resultado do estágio *decode/control* correspondente à mesma instrução deverá ocorrer num ciclo posterior *i+n*.

14) Na simulação, você deverá localizar com o auxílio dos endereços obtidos em **test.lst**:
    a. Início do main (opcode e endereço).
    b. Início do fibo (opcode e endereço).
    c. Escrita na memória do número x[seu num_aluno] (opcode e endereço).
    d. Protocolo de comunicação utilizado na escrita na memória do item c, isto é observar a relação entre os sinais de controle do processador-memória, a colocação do endereço de escrita e do dado a ser escrito.
15) Imprima o resultado da simulação, nos trechos de interesse. Anote o tempo de computação do número de fibonacci (sem a inicialização).

→ No relatório:

DADOS EXPERIMENTAIS:

Carta de tempos com simulação com trechos de interesse (legíveis) como descrito on item 14.

ANÁLISE E DISCUSSÃO:

1) **Indique todos os pontos de interesse como descrito on item 14 a, b e c, identificando os opcodes e endereços de memória da instrução correspondente.**
2) **Indique o ponto de interesse do item 14 d, identificando os sinais de controle, o endereço de escrita e o dado a ser escrito.**
3) **Explicar à luz de 14 d como é a sequência de eventos para uma escrita à memória.**
4) **Estime o tempo de computação do número de fibonacci por hardware (exp. 1B) para o mesmo número que você utilizou (num_aluno) e o compare criticamente com o medido por software-processador nesta experiência 2.**

APÊNDICE

Intel HEX-record Format (retirado de http://www.lucidtechnologies.info/)

INTRODUCTION

Intel's Hex-record format allows program or data files to be encoded in a printable (ASCII) format. This allows viewing of the object file with standard tools and easy file transfer from one computer to another, or between a host and target. An individual Hex-record is a single line in a file composed of many Hex-records.

HEX-RECORD CONTENT

Hex-Records are character strings made of several fields which specify the record type, record length, memory address, data, and checksum. Each byte of binary data is encoded as a 2-character hexadecimal number: the first ASCII character representing the high-order 4 bits, and the second the low-order 4 bits of the byte.

The 6 fields which comprise a Hex-record are defined as follows:

Field		Characters	Description
1	Start code	1	An ASCII colon, ":".
2	Byte count	2	The count of the character pairs in the data field.
3	Address	4	The 2-byte address at which the data field is to be loaded into memory.
4	Type	2	00, 01, or 02.
5	Data	0-2n	From 0 to n bytes of executable code, or memory loadable data. n is normally 20 hex (32 decimal) or less.
6	Checksum	2	The least significant byte of the two's complement sum of the values represented by all the pairs of characters in the record except the start code and checksum.

Each record may be terminated with a CR/LF/NULL. Accuracy of transmission is ensured by the byte count and checksum fields.

HEX-RECORD TYPES

There are three possible types of Hex-records.

00	A record containing data and the 2-byte address at which the data is to reside.
01	A termination record for a file of Hex-records. Only one termination record is allowed per file and it must be the last line of the file. There is no data field.
02	A segment base address record. This type of record is ignored by Lucid programmers.

HEX-RECORD EXAMPLE

Following is a typical Hex-record module consisting of four data records and a termination record.

```
:10010000214601360121470136007EFE09D2190140
:100110002146017EB7C20001FF5F16002148011988
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```

The first data record is explained as follows:

: Start code.

10 Hex 10 (decimal 16), indicating 16 data character pairs, 16 bytes of binary data, in this record.

0100 Four-character 2-byte address field: hex address 0100, indicates location where the following data is to be loaded.

00 Record type indicating a data record.

The next 16 character pairs are the ASCII bytes of the actual program data.

40 Checksum of the first Hex-record.

The termination record is explained as follows:

: Start code.

00 Byte count is zero, no data in termination record.

0000 Four-character 2-byte address field, zeros.

01 Record type 01 is termination.

FF Checksum of termination record.