Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Arquitetura de Computadores/Organização de computadores I - 2018/2

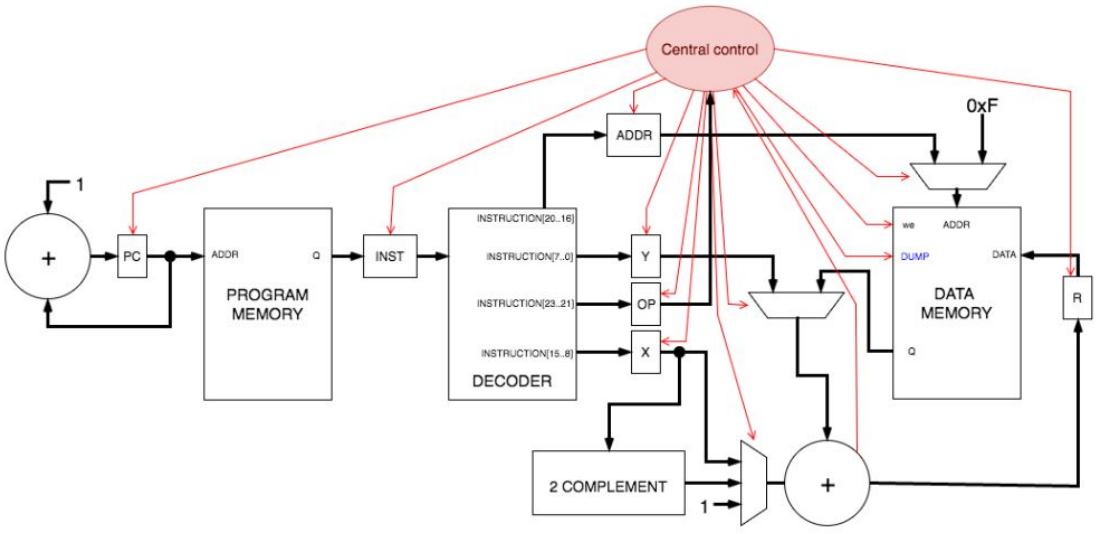
Trabalho a ser realizado em grupos de 4 alunos no máximo
Data de entrega: 21/11/2018

1. Descrição geral

O projeto consiste em implementar Nibble, um processador de 8 bits com conjunto de instruções reduzido, cujo núcleo foi projetado com arquitetura Harvard e 4 estágios de pipeline. Esse processador possui apenas 6 instruções: ADD, ADDP, SUB, SUBP, END e MDUMP. As instruções são gravadas na memória de programa (também conhecida como memória flash) e utiliza a memória RAM para armazenar os resultados das operações. O conteúdo do processador são os seguintes módulos:

1. Memória de programa (ROM);
2. Decodificador de instrução;
3. Memória de dados (RAM);
4. Multiplexadores;
5. Módulo complemento a 2;
6. Somador completo (8 bits);
7. Registradores de carga paralela (PC, ADDR, X, OP, Y e R) ;
8. Módulo de controle central.

Observação: Os módulos 1 e 3 serão fornecidos, não precisam ser desenvolvidos. Os outros módulos ficam a critério do aluno.



Atenção: A implementação do sistema será um pouco diferente da figura, devido ao pipeline.

1.1 INSTRUÇÕES

O processador Nibble possui 6 instruções: ADD, ADDP, SUB, SUBP, END e MDUMP. Todas as instruções tem o mesmo tamanho (24 bits), onde 3 bits são para código de operação e os outros 21 bits para operandos ou reservados. A descrição da cada instrução está detalhada nas tabelas a seguir:

Atenção: em todos os exemplos são usados números em binário e hexadecimal.

ADD

Operação	Endereço de Armazenamento	Literal	Literal
ADD	addr	x	y
A operação realiza a soma dos valores de x e y e armazena o resultado no endereço **addr** da memória RAM. **ADD addr, x, y => [addr] = x + y**			
000	aaaaa(5 bits)	xxxxxxxx(8 bits)	yyyyyyyy(8 bits)
Exemplo: 000.00110.00000111.00000001 <=> ADD 6, 7, 1 O endereço 0x6 da memória RAM receberá o valor 0x8.			

ADDP

Operação	Endereço de Armazenamento	Literal	Reservado
ADDP	addr	x	-
A operação realiza a soma dos valores de **x** e do conteúdo do endereço **addr** e armazena o resultado no endereço **addr** da memória RAM. **ADDP addr, y => [addr] = [addr] + x**			
001	aaaaa(5 bits)	xxxxxxxx(8 bits)	- - - - - - - - (8 bits)
Exemplo: 001.00010.00001001.00000000 <=> ADDP 2, 9 O conteúdo do endereço 0x2 da memória RAM será incrementado em 9.			

SUB

Operação	Endereço de Armazenamento	Literal	Literal
SUB	addr	x	y
A operação subtrai **x** de **y** e armazena o resultado no endereço **addr** da memória RAM. **SUB addr, y, x => addr = y - x**			
010	aaaaa(5 bits)	xxxxxxxx(8 bits)	yyyyyyyy(8 bits)
Exemplo: 010.01010.00000010.00001001 <=> SUB A, 2, 9 O conteúdo do endereço 0xA da memória RAM receberá o valor 7.			

SUBP

Operação	Endereço de Armazenamento	Literal	Reservado
SUBP	addr	x	-
A operação subtrai **x** do conteúdo de **addr** e armazena o resultado no endereço **addr** da memória RAM. **SUBP addr, y => [addr] = [addr] - x**			
011	aaaaa(5 bits)	xxxxxxxx(8 bits)	- - - - - - - - (8 bits)
Exemplo: 011.00010.00001001.00000000 <=> SUBP 2, 9 O conteúdo do endereço 0x2 da memória RAM será decrementado em 9 unidades.			

Observação: Todas as operações aritméticas podem gerar a exceção overflow. Caso isso ocorra, o somador coloca a saída ovf para o valor 1 indicando ao controle central o erro. A posição 0xF da memória RAM armazena um contador de overflow ocorridos durante a execução. Para cada overflow que ocorrer, o contador deve ser incrementado na posição 0xF da memória RAM.

END

Operação	Reservado
END	-
Essa instrução indica ao processador que o programa acabou.	
100	- (21 bits)
Exemplo: 100.00000.00000000.00000000 <=> END O Controle Central para de funcionar.	

MDUMP

Operação	Reservado
MDUMP	-
Ferramenta de depuração. Ao ser executada essa operação, o conteúdo da memória RAM será externalizado um a um na porta Q da memória. A implementação dessa instrução já está contemplada no código da memória RAM, basta o controle central indicar quando executar essa operação colocando o valor da entrada DUMP na memória para 1. Caso contrário o valor da entrada DUMP deve ser mantido em 0.	
111	- (21 bits)
Exemplo: 111.00000.00000000.00000000 <=> MDUMP	

1.2 MÓDULOS

Os módulos do processador estão descritos a seguir. Os módulos Memória ROM e Memória RAM serão fornecidos.

1.2.1 Somador completo

Módulo que implementa um somador completo de oito bits o qual contém detector de overflow. Caso haja um overflow a saída ovf é mantida 1 até que os operandos sejam alterados.

1.2.2 Complemento a 2

Esse módulo é responsável por fazer o complemento a 2 de um número de 8 bits que ele recebe na entrada. A saída do módulo tem 8 bits.

1.2.3 Decodificador de instruções

O módulo decodificador é responsável por separar o conteúdo da instrução em 4 partes:

- Código de operação: **OP[2..0] = INST[23..21]**
- Endereço: **ADDR[4..0] = INST[20..15]**
- Operando de 8 bits: **X[7..0] = INST[15..8]**
- Operando de 8 bits: **Y[7..0] = INST[7..0]**

1.2.4 Registradores de carga paralela

Os registradores PC, INST, ADDR, Y, X, OP, X e R são registradores de carga paralela. Eles possuem as seguinte entradas e saídas: **data_in** (8 bits), **clk** (1 bit), **en** (1 bit) e **data_out** (8 bits). A mudança do dado de saída é engatilhada na subida do clock. O valor da saída apenas será alterado se o pino **en** (enable) estiver habilitado; caso contrário, o valor da saída se mantém, independente das mudanças do clock.

1.2.5 Memória de programa

A memória de programa é somente leitura e contém o programa a ser executado pelo processador. Possui palavras de 24 bits, uma entrada **ADDR** de 8 bits e uma saída **Q** onde será externalizado o conteúdo (24 bits) do endereço colocado na entrada. O dado de saída será alterado quando o endereço de entrada for alterado. A mudança da saída é engatilhada na descida do clock.

1.2.6 Memória de dados

A memória de programa é de leitura e escrita. Tem como entrada o **ADDR** 5 bits (endereço de leitura/escrita), a flag **we** 1 bit (write_enable), a flag **DUMP** 1 bit reservada e a entrada **DATA** a qual recebe o valor a ser escrito na memória. A leitura é feita através da saída **Q** de 8 bits a qual é engatilhada na descida do clock e apenas será alterada caso a flag **we** esteja desabilitada e a entrada **ADDR** mude. Para escrita (engatilhada na descida do clock), o dado deve ser colocado na entrada **DATA**, o endereço de escrita deve ser colocado no **ADDR** e a flag **we** deve ser habilitada.

1.2.7 Controle Central

É o módulo responsável pelo funcionamento do sistema. Ele habilita os registradores, ajusta os multiplexadores e utiliza as memórias de forma harmônica. O funcionamento do Controle Central é baseado no conceito de pipeline onde podemos identificar 3 estágios independentes de funcionamento do processador: FETCH, DECODE, EXECUTE/STORE.

- **FETCH:** é quando o valor do PC é alterado e o conteúdo da memória de programa (instrução) fica disponível;
- **DECODE:** o registrador de instrução (INST) é alterado, deixando pronta a instrução decodificada;

- **EXECUTE:** quando os registradores ADDR, Y, OP e X recebem os dados da instrução e o controle central ajusta os multiplexadores para que a instrução seja executada de forma apropriada. Em algumas instruções deve-se ter acesso à memória antes de executar (aumentando o tempo para que a memória externalize o dado - lembre-se que a memória, ao contrário dos registradores, é engatilhada na descida). O resultado fica disponível para ser armazenado no registrador **R** para posterior armazenamento na memória;
- **STORE:** os dados do registrador **R** serão armazenados na memória de dados. Observe que o processador só poderá executar uma nova instrução quando o **STORE** terminar sua operação.

2. Observações do projeto

- A implementação dos módulos 1 e 3 (memória RAM e ROM) estão fornecidas, tendo sido implementadas no Quartus;
- A figura fornecida serve como um guia, mas note que o resultado final será um pouco diferente, por ter pipeline;
- Considerem a existência de *hazards* na implementação. Lembrem-se de tratá-los, caso encontrados;
- **Sugere-se fortemente** o uso das ferramentas Quartus Prime para compilação e ModelSim para simulação, disponíveis para download no site da Altera (a versão lite é gratuita). Existem outras ferramentas pagas ou livres na Internet, mas o Quartus e o ModelSim são mais adequadas para o nosso trabalho;
- A forma como os testes devem ser feitos (para atestar o funcionamento do sistema) é opção do grupo, mas estes **devem** existir.

3. Documentação

- Juntamente ao código fonte, deverá ser entregue uma documentação contendo todas as decisões de projeto que foram tomadas durante a implementação, sobre aspectos não contemplados na especificação, assim como uma justificativa para essas decisões.
- **A documentação deve conter no máximo dez páginas.**
- A documentação deve indicar o nome dos alunos. O código fonte não deve ser incluído no arquivo PDF da documentação.
- **É obrigatório apresentar:**

- Os **testes** que demonstram o correto funcionamento do código desenvolvido;
- Uma **imagem do sistema atualizada** (semelhante a apresentada nessa proposta, porém com pipeline e sinais de controle adicionados)

4. Informações Importantes

- O trabalho deve ser feito em grupos de até 4 alunos, podendo ser discutido entre os colegas desde que não haja cópia ou compartilhamento do código fonte. **Caso detectado, será atribuído nota zero a todos os alunos envolvidos no plágio.**
- A data de entrega será especificada através de uma tarefa no Moodle.
- Os trabalhos poderão ser entregues até às 23:55 do dia especificado para a entrega.
- O horário de entrega deve respeitar o relógio do sistema Moodle. Haverá uma tolerância de 5 minutos de atraso, de forma que os alunos podem fazer a entrega até às 0:00. A partir desse horário, os trabalhos já estarão sujeitos a penalidades. A fórmula para desconto por atraso na entrega do trabalho prático é:

$$desconto = \frac{2^d}{0.32}\%,$$

onde d é o atraso em dias úteis. Note que após 5 dias úteis, o trabalho não pode ser mais entregue.

- Todas as dúvidas referentes ao trabalho serão esclarecidas por meio do fórum disponível no ambiente Moodle da disciplina.
- A entrega do trabalho deverá ser realizada pelo Moodle, na tarefa criada especificamente para tal. Deverá ser entregue um único arquivo compactado em formato ".zip" com nome "tp2_**NomeESobrenome**DoAluno.zip", contendo todo o código fonte, e a documentação em formato PDF.
- **Trabalhos que descumprirem os padrões definidos acima serão penalizados.**

Referências

https://www.intel.com/content/www/us/en/programmable/support/training/university/materials-tutorials.html

https://www.embarcados.com.br/tutorial-de-verilog-projeto-com-quartus/

https://www2.pcs.usp.br/~labdig/material/GuiaResumido-Quartus_II_91_corrigida.pdf

http://www.dca.fee.unicamp.br/~lboccato/tutorial_quartus_9p1_v02_Q2_20145.pdf