INSTITUTO FEDERAL
Paraná
Campus Pinhais

Bacharelado em ciência da computação
Arquitetura e sistemas operacionais
Trabalho 1: circuito de processador, Mico — 3 de novembro de 2024

Instruções para entrega dos exercícios

- O trabalho pode ser feito em duplas. Não esqueça de identificar corretamente os membros da dupla.
- Plágio não será tolerado.
- Utilize o simulador de circuitos Digital para implementar as soluções.
- Apenas um integrante do grupo deve entregar a solução no Classroom.

1 O processador

O seu trabalho é implementar, usando o simulador de circuitos Digital, o circuito de ciclo longo que vimos em sala. Vamos chamar o nosso modelo de processador de Mico.

O Mico possui um conjunto reduzido de instruções, com apenas 16 delas, conforme a tabela 1. A tabela apresenta cada instrução e seu respectivo código de identificação, chamado *opcode*. A coluna *semântica* indica o comportamento da instrução.

A separação por vírgula na coluna *semântica* indica comportamentos simultâneos: na instrução de soma, o registrador apontado por `c` tem seu valor atualizado com a soma dos valores nos registradores `a` e `b`; **ao mesmo tempo**, o apontador de instrução (IP) é acrescido de 1, para que o processador execute a próxima instrução quando um novo ciclo de relógio se iniciar. Note que a instrução `j const` é uma pseudo-instrução, implementada usando a instrução `jal`, com *opcode* `c`.

Tabela 1: Conjunto de instruções do Mico.

opcode	instrução	semântica	descrição
0	`add c, a, b`	$R(c) \leftarrow R(a) + R(b), IP \leftarrow IP + 1$	soma
1	`sub c, a, b`	$R(c) \leftarrow R(a) - R(b), IP \leftarrow IP + 1$	subtração
2	`mul c, a, b`	$R(c) \leftarrow R(a) * R(b), IP \leftarrow IP + 1$	multiplicação
3	`and c, a, b`	$R(c) \leftarrow R(a) \wedge R(b), IP \leftarrow IP + 1$	conjunção
4	`or c, a, b`	$R(c) \leftarrow R(a) \vee R(b), IP \leftarrow IP + 1$	disjunção
5	`xor c, a, b`	$R(c) \leftarrow R(a) \oplus R(b), IP \leftarrow IP + 1$	ou-exclusivo
6	`slt c, a, b`	$R(c) \leftarrow (R(a) < R(b)), IP \leftarrow IP + 1$	*set on less than*
7	`not c, a`	$R(c) \leftarrow not(R(a)), IP \leftarrow IP + 1$	complemento
8	`addi c, a, b`	$R(c) \leftarrow R(a) + extSin(const), IP \leftarrow IP + 1$	soma constante aritmética
9	`ld c, const(a)`	$R(c) \leftarrow M[R(a) + extSin(const)], IP \leftarrow IP + 1$	*load from memory*
a	`st const(a), b`	$M[R(a) + extSin(const)] \leftarrow R(b), IP \leftarrow IP + 1$	*store to memory*
b	`show a`	$display \leftarrow R(a), IP \leftarrow IP + 1$	exibe valor na saída
c	`j const`	$IP \leftarrow const, R(0) \leftarrow IP + 1$	salto incondicional
c	`jal c, const`	$IP \leftarrow const, R(c) \leftarrow IP + 1$	*jump and link*
d	`jr a`	$IP \leftarrow R(a)$	*jump register*
e	`beq a, b, const`	$IP \leftarrow ((R(a) == R(b)?const : IP + 1))$	desvio condicional
f	`halt`	$IP \leftarrow IP + 0$	paralisa a execução

Todas as instruções possuem 32 bits e o mesmo formato, como mostra a figura 1.

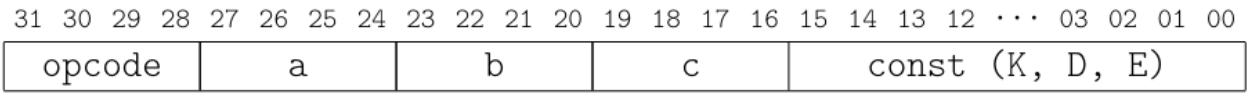

Figura 1: Formato das instruções do Mico.

2 Componentes do processador

O seu processador deve conter alguns componentes que serão descritos a seguir.

Um *instruction pointer* (IP): um registrador de 16 bits que aponta o endereço da próxima instrução a ser executada.

Uma memória de instruções (MI): memória ROM com capacidade para 64K palavras, para armazenar o programa a ser executado. Cada palavra dessa memória será uma instrução que, como visto anteriormente, tem 32 bits.

Um circuito de controle: o circuito deve gerar todos os sinais de controle do processador. O jeito mais fácil de implementar esse componente é usando uma memória ROM com 16 palavras de C bits (gerando C sinais de controle). Cada uma das 16 palavras será usada por exatamente uma das instruções.

Somador: para incrementar o IP.

Multiplexadores: para escolher uma das entradas possíveis.

ULA: unidade de lógica e aritmética, com operandos e resultados de 32 bits.

Extensor de sinal: para estender a constante de 16 bits para 32 bits, considerando o complemento de dois.

Banco de registradores (R): bloco com 16 registradores, cada um com 32 bits de largura. O registrador zero sempre deve conter o valor zero quando lido. Escritas no registrador zero devem ser ignoradas.

Memória de dados (MD): memória RAM com capacidade para 64K palavras de 32 bits, para armazenar os dados do programa.

A figura 2 mostra o circuito do Mico. Note que, para simplificar o desenho, alguns sinais são apelidados e aparecem "magicamente" em outra parte do circuito. Nesses casos, o seu circuito no simulador deve contemplar um sinal (um fio) que irá atravessar todo o desenho. Organize os sinais do seu circuito para que seja compreensível!

3 O que fazer

Você deve entregar um arquivo compactado (no formato *.zip*), cujo nome é formado pelas iniciais dos integrantes do grupo, separados por espaço. Por exemplo, o grupo formado por Fulano da Silva e Beltrano José de Souza deve entregar um arquivo com o nome *fs-bjs.zip*.

Ao ser descompactado, seu arquivo deve gerar uma pasta, também com as iniciais dos integrantes (seguindo o exemplo, *fs-bjs*). Dentro desse diretório, **deve** existir um arquivo *README.md* que contenha o nome completo dos integrantes e outras informações que o grupo achar relevante sobre o trabalho.

Além disso, a pasta do trabalho deve conter um arquivo chamado *mico.dig* com o circuito do processador. Outros arquivos *.dig* devem existir para garantir que o circuito possa ser compreendido.

Crie pelo menos o seguinte: um arquivo *.dig* para a ULA do seu projeto e um arquivo *.dig* para o banco de registradores.

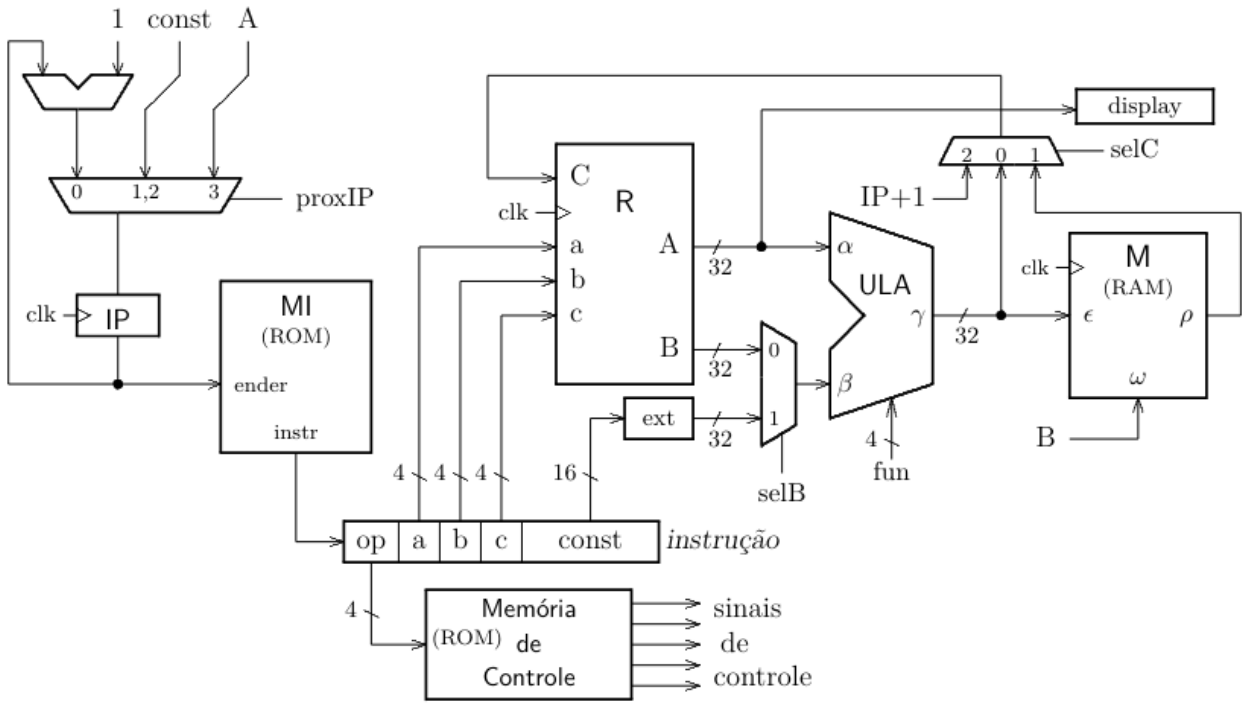

Figura 2: Circuito de dados e controle do Mico.

3.1 Componentes do simulador Digital

Abaixo segue uma descrição de quais componentes oferecidos pelo simulador podem ser usados por você na construção do seu processador. Se um componente não foi explicitamente permitido, então ele é proibido e não deve ser usado. O uso de componentes não permitidos do simulador acarretará em desconto no conceito do trabalho.

Na seção *Lógica*, você pode usar todas as portas lógicas (todos os componentes da seção exceto *LookUpTable*).

Na seção *Entrada e saída*, você deve usar os componentes chamados *Entrada* e *Saída* para indicar as entradas e saídas do seu circuito e dos circuitos que o formam. Você deve usar um (e apenas um) componente *Entrada do Clock* que será o relógio do seu circuito, sincronizando o funcionamento. Você pode usar o componente *LED* e o *Gráfico de dados* para auxiliar na inspeção dos sinais e da execução do seu processador.

Na seção *Conexões*, você pode usar *Valor constante* quando precisar de um número fixo nas suas operações, e o *Distribuidor* para aglomerar vários bits em um só sinal ou para desmembrar um sinal de vários bits em vários sinais.

Na seção *Plexers*, você pode usar o *Multiplexador*, *Demultiplexador* e o *Decodificador*, de acordo com a necessidade.

Na seção *Flip-Flops*, você pode usar o *Flip-Flop D*.

Na seção *Memória*, você pode usar o *Registrador* (que é basicamente um Flip-Flop D com um sinal de *enable*) e a *ROM*. Na subseção *RAM*, você pode usar a *RAM, portas separadas* para a MD.

Na seção *Aritmética*, você pode usar *Somar*, *Multiplicar*, *Comparador* e *Extensor de sinal*.

Você pode usar os elementos em *Diversos* → *Decoração* para auxiliar a organizar seu circuito.

3.2 (Sub)circuitos obrigatórios

Como dito acima, você entregará um arquivo contendo o circuito para o processador e deverá criar subcircuitos, ao menos para a ULA e para o banco de registradores. O circuito principal deve incluir os "componentes" da

ULA e do banco de registradores usando a seção *Personalizar*.

3.2.1 A ULA

A sua ULA deve receber como entrada os operandos `A` e `B`, cada um com 32 bits, e o sinal `F` de 3 bits que indica qual operação a ULA deve mostrar na saída. A saída da ULA deve ser um sinal `S` de 32 bits, além do sinal de um bit que indica se houve ou não *overflow*.

Note que o componente *Subtrair* do simulador não pode ser utilizado. Você deve usar o componente *Somar* para realizar a subtração, da mesma forma que fez na lista de exercícios da ULA.

Você pode complementar a ULA para gerar mais sinais, se achar necessário. Suas alterações, obviamente, não devem atrapalhar o funcionamento descrito nessa especificação.

3.2.2 O banco de registradores

O banco de registradores deve receber 3 entradas de 4 bits indicando quais são os registradores a serem acessados (`a, b, c`). Um sinal `wr` de entrada deve habilitar (ou não) a escrita no registrador `c`. O sinal de entrada de 32 bits `C` contém o valor que será armazenado em `c`. Seu banco de registradores também deve receber um sinal de entrada que será o *clock*.

Seu banco de registradores deve conter 16 registradores, sendo que o registrador zero ignora escritas (mesmo com $wr = 1$) e possui sempre o valor 0 (zero) armazenado. Por fim, seu banco de registradores deve gerar duas saídas de 32 bits, A e B, com o conteúdo dos registradores lidos.

3.3 Controle

O seu circuito (ou tabela, usando ROM) de controle deve gerar todos os sinais necessários para o bom funcionamento do processador, como visto em sala. Isso inclui, mas não se limita, a:

1. sinal que escolhe o próximo IP
2. sinal que habilita escrita no registrador de destino
3. sinal que escolhe a entrada B da ULA
4. sinal que escolhe o resultado C para escrever no registrador
5. sinal que escolhe a operação da ULA
6. sinal que habilita a escrita na MD

Você é o responsável por identificar outros sinais necessários e também o tamanho de cada sinal. Utilize as informações de sala de aula e do livro de referência para te auxiliar nas escolhas!

3.4 Outras considerações

O seu circuito deve funcionar! Isso significa que você deve testar o circuito completo e garantir que ele gera os resultados corretos.

Você pode preencher a memória de instruções com um programa (em binário), de forma que o seu processador executará elas. Crie programas de teste fazendo operações na MD e verificando os resultados.

Sugiro que você use um registrador a mais para armazenar a última informação mostrada pela instrução `show`. Isso pode facilitar muito os seus testes.