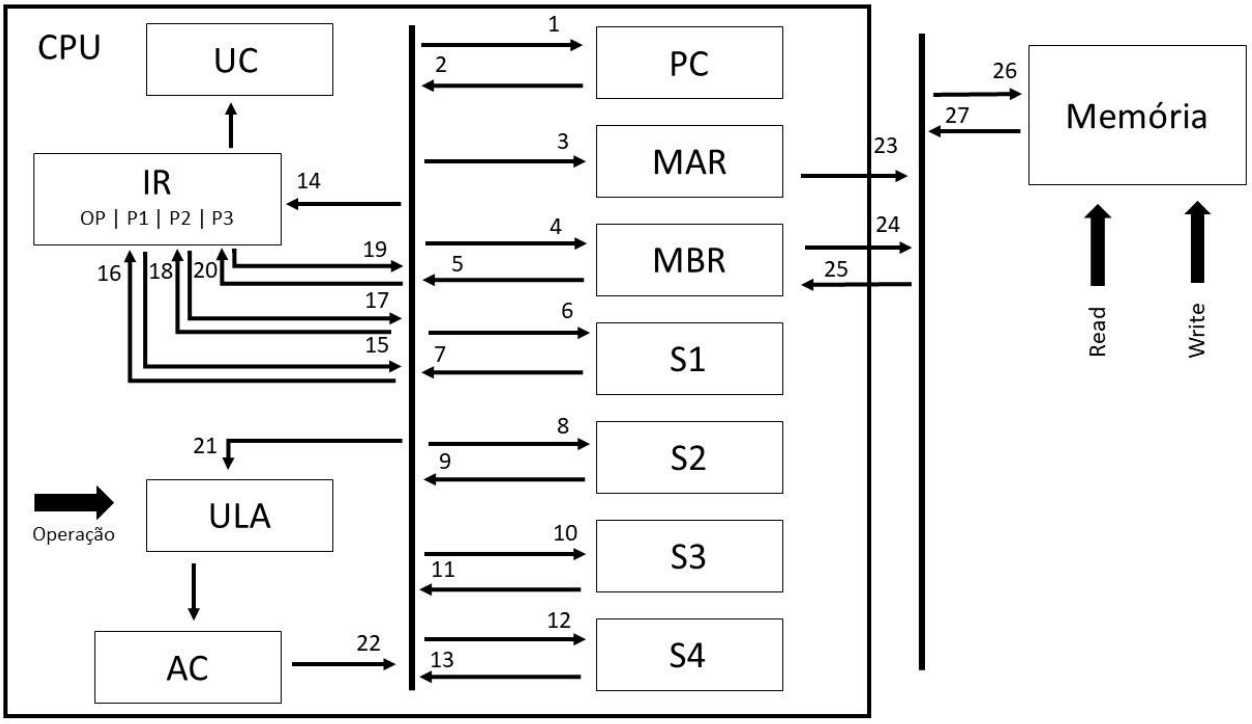
Relatório EP2

ACH2034 - Organização de Computadores Digitais - Turma 94

Alexandre Kenji Okamoto	11208371
Karina Duran Munhos	11295911
Matheus Antonio Cardoso Reyes	11270910
Mirela Mei Costa	11208392

1 ESPECIFICAÇÕES

1.1 Arquitetura de hardware



1.2 Código de máquina

1.2.1 Composição Total da Palavra (40 bits)

Sinais de Controle + OP ULA + OP Memória + Condição de pulo + Endereço da condição de pulo

1.2.2 Opcodes

1.2.2.1 Registradores

- $S1: 001

- $S2: 010
- $S3: 011
- $S4: 100

1.2.2.2 Memória
- Não ler nem escrever: 00
- Read: 01
- Write: 10

1.2.2.3 ULA
- nenhuma operação: 0000
- li: 0001
- lw: 0010
- sw: 0011
- move: 0100
- add: 0101
- sub: 0110
- beq: 0111
- bne: 1000
- j: 1001
- slt: 1010
- la: 1011

1.2.2.4 Condição de pulo
- Não pule: 000
- Pule: 001
- Pule se a subtração for igual 0: 010
- Pule se a subtração for diferente de 0: 011
- Pule se a subtração for positiva: 100
- Pule se a subtração for negativa: 101

1.2.3 Limites para tamanhos de constantes ou endereços

O número máximo é de 12 bits.

1.3 Microprograma

1.3.1 Ciclo de busca

t1: MAR <- PC t1: 01100000000000000000001000000 0000 00 000 0000
ULA <- PC
t2: memória <- MAR t2: 00000000000000000000000010010 0000 10 000 0000

t3: MBR <- memória t3: 10000000000000000000100101 0000 01 000 0000
PC <- ULA
t4: IR <- MBR t4: 00001000000001000000000000 0000 00 000 0000

1.3.2 Ciclos de execução

ADD R1, R2, R3 (REGISTRADOR COM REGISTRADOR)
t1: MAR <- IRend t1: 00100000000000000000000000 0000 00 000 0000
t2: ULA <- R3 t2: 00000000000000000001000000 0000 00 000 0000
t3: ULA <- R2 t3: 00000000000000000001000000 0101 00 000 0000
t4: R1 <- ULA t4: 00000000000000000000100000 0000 00 000 0000

ADD R1, R2, C (REGISTRADOR COM CONSTANTE)
t1: MAR <- IRend t1: 00100000000000000000000000 0000 00 000 0000
t2: ULA <- IR (p2 = C) t2: 00000000000000000001000000 0000 00 000 0000
t3: ULA <- R2 t3: 00000000000000000001000000 0101 00 000 0000
t4: R1 <- ULA t4: 00000000000000000000100000 0000 00 000 0000

SUB R1, R2, R3 (REGISTRADOR COM REGISTRADOR)
t1: MAR <- IRend t1: 00100000000000000000000000 0000 00 000 0000
t2: ULA <- R3 t2: 00000000000000000001000000 0000 00 000 0000
t3: ULA <- R2 t3: 00000000000000000001000000 0110 00 000 0000
t4: R1 <- ULA t4: 00000000000000000000100000 0000 00 000 0000

SUB R1, R2, C (REGISTRADOR COM CONSTANTE)
t1: MAR <- IRend t1: 00100000000000000000000000 0000 00 000 0000
t2: ULA <- IR (p2 = C) t2: 00000000000000000001000000 0000 00 000 0000
t3: ULA <- R2 t3: 00000000000000000001000000 0110 00 000 0000
t4: R1 <- ULA t4: 00000000000000000000100000 0000 00 000 0000

MOVE R1, R2
t1: MAR <- IRend t1: 00100000000000000000000000 0000 00 000 0000
t2: R1 <- R2 t2: 00000000000000000000000000 0100 00 000 0000

BEQ R1, R2, E (REGISTRADOR COM REGISTRADOR)
t1: MAR <- IRend t1: 00100000000000000000000000 0000 00 000 0000
t2: ULA <- R1 t2: 00000000000000000001000000 0000 00 000 0000
t3: ULA <- R2 t3: 00000000000000000001000000 0111 00 010 0000

BEQ R1, C, E (REGISTRADOR COM CONSTANTE)
t1: MAR <- IRend t1: 00100000000000000000000000 0000 00 000 0000
t2: ULA <- IR (p2 = C) t2: 00001000000000000001000000 0000 00 000 0000

t3: ULA <- R1 t3: 00000000000000000001000000 0111 00 010 0000

BNE R1, R2, E (REGISTRADOR COM REGISTRADOR)
t1: MAR <- IRend t1: 00100000000000000000000000 0000 00 000 0000
t2: ULA <- R1 t2: 00000000000000000001000000 0000 00 000 0000
t3: ULA <- R2 t3: 00000000000000000001000000 1000 00 011 0000

BNE R1, C, E (REGISTRADOR COM CONSTANTE)
t1: MAR <- IRend t1: 00100000000000000000000000 0000 00 000 0000
t2: ULA <- IR (p2 = C) t2: 00000000000000000001000000 0000 00 000 0000
t3: ULA <- R1 t3: 00000000000000000001000000 1000 00 011 0000

SLT R1, R2, R3
t1: MAR <- IRend t1: 00100000000000000000000000 0000 00 000 0000
t2: ULA <- R3 t2: 00000000000000000001000000 0000 00 000 0000
t3: ULA <- R2 t3: 00000000000000000001000000 1010 00 000 0000
t4: R1 <- ULA t4: 00000000000000000000100000 0000 00 000 0000

LI R1, C
t1: MAR <- IRend t1: 00100000000000000000000000 0000 00 000 0000
t2: R1 <- IR (p2 = C) t2: 00000000000000010000000000 0000 00 000 0000

J E
t1: MAR <- IRend t1: 00100000000000000000000000 0000 00 000 0000
t2: PC <- IR(E) t2: 10000000000000000000000000 1001 00 001 0000

LA R1, E
t1: MAR <- IRend t1: 00100000000000000000000000 0000 00 000 0000
t2: MBR <- memória t2: 00000000000000000000000101 0000 01 000 0000
t3: R1 <- MBR t3: 00001000000000000000000000 0001 00 000 0000

SW R1, C, R2
t1: MAR <- IRend t1: 00100000000000000000000000 0000 00 000 0000
t2: ULA <- IR(p2 = C) t2: 00000000000000010001000000 0101 00 000 0000
t3: ULA<- R2 t3: 00000000000000000101000000 0101 00 000 0000
t4: MAR <- ULA t4: 00100000000000000000100000 0000 00 000 0000
t5: MBR <- R1(p1) t5: 00010000000001000000000000 0000 00 000 0000
t6: memória <- MAR t6: 00000000000000000000010010 0000 01 000 0000
t7:memória<- MBR t7: 00000000000000000000001010 0000 10 000 0000

LW R1, C, R2
t1: MAR <- IRend t1: 00100000000000000000000000 0000 00 000 0000

t2: MBR <- memória t2: 00000000000000000000000101 0000 01 000 0000
t3: ULA <- MBR t3: 00001000000000000000001000000 0000 00 000 0000
t4: ULA <- R2 t4: 00000000000000000000001000000 0010 00 000 0000
t5: R1 <- ULA t5: 00000000000000000000000100000 0000 00 000 0000

2 COMPILAÇÃO

Para compilar basta usar o “javac *.java”. Não é necessário nenhum parâmetro especial. A classe principal é a “UC.java”.

Passos que fizemos para gerar um executável:

1. Ter os arquivos .java e .class
2. Criar um .txt chamado “manifest” com “Main-Class: UC” e 2 enters (no total 3 linhas)
3. No terminal digitar “jar cfm Programa.jar manifest.txt UC.class Microprograma.class Memoria.class FuncoesULA.class FuncoesAuxiliares.class”
4. Criar um .bat chamado “Programa” com “java -jar “Programa.jar””

3 EXECUÇÃO

1. Antes de rodar o programa, deve-se editar o arquivo “codigoMIPS.txt” (que está em branco) colando nele o código MIPS e salvar.
2. Dar duplo clique no arquivo “Programa.bat”, irá abrir o terminal e rodar o programa
3. Apertar a tecla “enter” para avançar
4. Se não der certo, executar do terminal normal mesmo, “java UC”.

4 CÓDIGO

Observações: o código recebe todos os comandos em letras minúsculas e os comandos de pulo (bne, beq e j) desconsideram linhas em branco e linhas com as palavras .word, .text e .data; apenas as linhas com operações entram no cálculo e a contagem de linha se inicia com 0, dá para observar em nossos testes.

4.1 UC.java (main)

É a classe principal que gerencia e chama as outras classes quando necessário. Recebe o arquivo txt com o código MIPS e realiza as operações necessárias de cada linha.

4.2 Memoria.java

Classe destinada à memória.

Memoria: construtor que instancia o *endereco* e o *conteudo*.

getEndereco: retorna o *endereco*.

getConteudo: retorna o *conteudo*.

setConteudo: recebe como parâmetro o conteúdo e atribui ao *conteudo*.

4.3 Microprograma.java

Classe destinada ao microprograma.

Microporgrama: construtor que instancia o *endereco* e o *conteudo*.

setConteudo: recebe como parâmetro o conteúdo e atribui ao *conteudo*.

getEndereco: retorna o *endereco*.

getConteudo: retorna o *conteudo*.

4.4 FuncoesULA.java

Classe que cuida das operações da ULA.

sw: recebe como parâmetro o código de máquina e armazena o valor indicado na posição do array indicada.

lw: recebe como parâmetro o código de máquina, armazena no CAR a primeira linha do ciclo de execução do *add*, chama o *exibirCicloExecucao* e armazena no registrador indicado o valor que está na posição indicada do array.

la: recebe como parâmetro o código de máquina e armazena no registrador indicado a constante indicada.

li: recebe como parâmetro o código de máquina, armazena no CAR a primeira linha do ciclo de execução do *add*, chama o *exibirCicloExecucao* e armazena no registrador indicado a constante indicada.

move: recebe como parâmetro o código de máquina, armazena no CAR a primeira linha do ciclo de execução do *add*, chama o *exibirCicloExecucao* e armazena no registrador indicado a constante indicada.

j: recebe como parâmetro o código de máquina e retorna a linha indicada, em decimal.

slt: recebe como parâmetro o código de máquina, armazena no CAR a primeira linha do ciclo de execução do *slt*, chama o *exibirCicloExecucao* e armazena no registrador indicado o resultado da comparação, 0 ou 1.

beq: recebe como parâmetro o código de máquina e a posição atual na memória, armazena no CAR a primeira linha do ciclo de execução do *beq* registrador com registrador ou registrador com constante, dependendo do que estiver indicado, chama o *exibirCicloExecucao* e retorna a mesma posição ou a posição indicada, dependendo do resultado da comparação.

bne: recebe como parâmetro o código de máquina e a posição atual na memória, armazena no CAR a primeira linha do ciclo de execução do *bne* registrador com registrador ou registrador com constante, dependendo do que estiver indicado, chama o *exibirCicloExecucao* e retorna a mesma posição ou a posição indicada, dependendo do resultado da comparação.

add: recebe como parâmetro o código de máquina, armazena no CAR a primeira linha do ciclo de execução do *add* registrador com registrador ou registrador com constante, dependendo do que estiver indicado, chama o

exibirCicloExecucao, e armazena no registrador indicado o resultado da soma.

sub: recebe como parâmetro o código de máquina, armazena no CAR a primeira linha do ciclo de execução da *sub* registrador com registrador ou registrador com constante, dependendo do que estiver indicado, chama o *exibirCicloExecucao*, e armazena no registrador indicado o resultado da subtração.

4.5 FuncoesAuxiliares.java

Classe para as funções auxiliares utilizadas no programa.

encontrarPosicaoMemoria: recebe como parâmetro um endereço na memória e retorna a posição na lista da memória correspondente.

exibirCicloExecução: imprime o ciclo de execução e atualiza o microprograma com as portas alteradas, recebe como parâmetro a linha de ínicio dele e o código de máquina da operação.

retornaPortaRegistrador: recebe como parâmetro um registrador e retorna suas portas de entrada e saída.

alterarPortas: recebe como parâmetro as portas e a posição que deve ser ligada.

armazenarLista: responsável por ler a linha de declaração do array e armazenar na memória.

lerMicroprograma: responsável por ler o microprograma e armazenar os dados.

verificaRegistrador: verifica se há um registrador de acordo com o opcode.

converterRegistrador: retorna o valor do registrador de acordo com o opcode.

converterLinguagemMontagem: recebe como parâmetro uma linha de código MIPS e converte para linguagem de montagem.

converterLinguagemMaquina: recebe como parâmetro a linguagem de montagem e converte para linguagem de máquina.

completarBinario: completa o número com zeros se o número ocupar menos que 4 bits.

completarBinarioDireita: completa o número com zeros a direita se o número ocupar menos que 4 bits.

isInteger: verifica se o número é um inteiro.

armazenarMemoria: armazena os dados na memória.

4.6 microprograma.txt

Contém o Ciclo de Busca e os Ciclos de Execução de cada comando em MIPS.

5 TESTES

Os testes que fizemos estão na pasta “Testes”. A organização dos testes é feita pelo nome da imagem: “Arquivo”[número do teste]”-”[Linha que está executando][instrução que está sendo executada]. Exemplo: “Arquivo3-2La.jpg”. E

cada screenshot é a execução de uma linha de código MIPS.

6 PROBLEMAS

1. Conversões de números binários negativos para números decimais não condizem com o resultado esperado, os registradores exibem resultados diferentes, -1 em binário por exemplo, é exibido nos registradores como o número decimal 4095, conversões de números binários positivos funcionam normalmente, uma possível solução é tratar os números binários negativos de forma individual, realizando seu complemento de 2 antes de realizar a conversão. Esse erro é apresentado nos arquivos 9, 10 e 11.
2. Os registradores não são atualizados e exibidos a cada tempo do microprograma, e sim a cada linha de um código em MIPS, uma maneira de solucionar isso é efetuar os cálculos e atualizações nos registradores a cada vez que um micro-instrução é gerada, em vez de cada linha de código MIPS.
3. Por consequência do problema anterior, não é exibido no programa os dados que passam pelo barramento interno e externo, isso ocorre pois deveria ser uma atualização a cada tempo do microprograma, ao solucionar o problema anterior, seria possível solucionar este.

7 FONTES

Função que verifica se é um inteiro. Retirada de https://stackoverflow.com/questions/5439529/determine-if-a-string-is-an-integer-in-java

FuncoesAuxiliares.java

```java
public static boolean isInteger(String s, int radix) {
    if(s.isEmpty()) return false;
    for(int i = 0; i < s.length(); i++) {
        if(i == 0 && s.charAt(i) == '-') {
            if(s.length() == 1) return false;
            else continue;
        }
        if(Character.digit(s.charAt(i),radix) < 0) return false;
    }
    return true;
}
```