**Instruções**

**Sobre as instruções**

As instruções são de 8 bits e os 4 bits mais significativos representam o código da instrução, os 4 bits menos significativos podem conter os parametros especificos da instrução.

Algumas instruções tem parametro na palavra seguinte, por exemplo:

**LOAD R, 255**

O endereço 255 não cabe nos 4 bits menos significativos, no entanto, eles são suficientes para indicar o primeiro operando (R). Assim, o endereço 255 fica na próxima instrução e serve como parametro. Quando isso acontece, o PC é incrementado mais uma vez.

No detalhamento das instruções, estamos enumerando os bits da direita para a esquerda e começando do 0, ou seja, o bit menos significativo é o 0 e o mais significativo é o 7.

**Detalhamento das instruções**

“0000” - NOP

Não faz nada.

“0001” - LOAD Reg1, addr

Carrega o valor do endereço addr em Reg1.

Parametros:

1. “0000” Reg1 = A,
2. “0001” Reg1 = B,
3. “0010” Reg1 = R;
4. próxima palavra indicará o endereço (addr)

Observação importante: o endereço deve ser um número **DECIMAL** de 0 a 255.

“0010” - CMP Reg1, Reg1

Compara Reg1 e Reg2, atualizando os registradores de flags. Reg1 e Reg2 em vez de registradores, podem ser a próxima palavra. Se forem iguais, o sinal de controle “equal\_flag” vai receber ‘1’, diferentes ‘0’. Se Reg1 é menor que Reg2, o sinal de controle “sign\_flag” recebe ‘1’, caso contrário, recebe ‘0’.

Parametros:

1. Bits 3 e 2: indicam o primeiro operando
   1. “00” se for A
   2. “01” se for B
   3. “10” se for R
   4. “11” se for a próxima palavra
2. Bits 1 e 0 indicam o segundo operando e seguem a mesma lógica acima.

Obs: não é permitido operações do tipo CMP A, A, onde os dois operandos são os mesmos registradores.

“0011” - JMP addr

Salta incondicionalmente para o endereço que estiver na próxima palavra.

Parametros:

1. addr: endereço para o salto (estará na palavra seguinte ao código da instrução).

“0100” - JEQ addr

Salta para o endereço da próxima palavra caso o resultado da última comparação seja igual.

Parametros:

1. addr: endereço para o salto (estará na palavra seguinte ao código da instrução).

“0101” - JGR addr

Salta para o endereço da próxima palavra caso na última comparação o primeiro operando era maior que o segundo.

Parametros:

1. addr: endereço para o salto (estará na palavra seguinte ao código da instrução).

“0110” - STORE Reg1, addr

Armazena o valor de Reg1 no endereço addr

Parametros:

1. “0000” - Reg1 será o registrador A
2. “0001” - Reg1 será o registrador B
3. “0010” - Reg1 será o registrador R
4. addr - será o endereço onde será salvo o valor do registrador. Este endereço estara na palavra seguinte à instrução.

“0111” - MOV Reg1, Reg2

Move o valor de Reg2 para Reg1. O valor Reg2 não precisa necessariamente estar em um registrador, ele pode estar na palavra seguinte à instrução.

Parametros:

1. Bits 3 e 2: vao definir o primeiro operando
   1. “00” - Reg1 será A
   2. “01”- Reg1 será B
   3. “10”- Reg1 será R
2. Bits 1 e 0: vão definir o segundo operando
   1. “00” - Reg2 será A
   2. “01”- Reg2 será B
   3. “10”- Reg2 será R
   4. “11” - Reg2 será o valor da próxima palavra

Obs: Não é permitido operações do tipo MOV B, B, ou seja, operações nas quais os dois operandos sejam iguais.

“1000” - ADD | “1001” - SUB | “1010” - AND | “1011” - OR | “1100” - NOT

Essas instruções utilizam a ULA e foram implementadas de maneira similar.

Parametros:

1. Bits 3 e 2: indicam o primeiro operando
   1. “00” primeiro operando será o registrador A.
   2. “01” primeiro operando será o registrador B
   3. “10” primeiro operando será o registrador R
   4. “11” primeiro operando será o valor na palavra seguinte
2. Bits 1 e 0: indicam o segundo operando e seguem a mesma lógica acima.

Depois de selecionados os operandos, eles serão colocados como entrada para a ULA e o resultado será salvo no registrador R.

“1101” - IN Reg1

Lê o valor das chaves do FPGA e armazena em Reg1

Será implementado posteriormente

“1110” - OUT Reg1

Exibe o valor de Reg1 nos LEDs da FPGA.

Será implementado posteriormente

“1111” - WAIT

Espera que um botão da FPGA seja pressionado para continuar.

Será implementado posteriormente

**Assembler**

1

Foi desenvolvido um assembler compatível com a CPU apresentada. Ele suporta rótulos para referências a endereços de memória, instruções básicas de controle, operações aritméticas/lógicas e comentários no código iniciados com “;”. O assembler inclui verificações de sintaxe e retorna erros explicativos caso ocorram.

## **Uso do Assembler**

### **Execução**

1. Certifique-se de que o Python está instalado em seu sistema.
2. Execute o código do assembler, ele estará no diretorio “assembler” em um arquivo chamado ***assembler.py***
3. Execute o assembler com o seguinte comando:

| python assembler.py <arquivo\_de\_entrada> <arquivo\_de\_saida> |
| --- |

#### **Parâmetros**

* <arquivo\_de\_entrada>: Nome do arquivo de entrada contendo o código assembly com a extensão .asm.
* <arquivo\_de\_saida>: Nome do arquivo de saída onde o código de máquina será salvo com a extensão .hex.

### **Formato do Código Assembly**

* Cada linha contém uma instrução ou rótulo.
* Rótulos devem terminar com **:**
* Comentários iniciam com **;** e são ignorados pelo assembler.
* Operandos de instruções são separados por espaços ou vírgulas.

#### **Exemplo:**

| ; Exemplo de código assembly START:  LOAD A, 10 ; Carrega o valor que está no end 10 no registrador A  CMP A, B ; Compara os valores de A e B  JEQ END ; Salta para END se A for igual a B  ADD A, B ; Soma A com B END:  STORE A, 20 ; Armazena o valor de A no endereço 20  NOP ; Nenhuma operação |
| --- |

## **Conjunto de Instruções**

### **Instruções Suportadas**

As instruções suportadas são as instruções apresentadas anteriormente, porém, aqui está uma tabela resumida das instruções. Caso seja necessário, verifique o detalhamento das instruções na seção anterior.

| **Instrução** | **Opcode** | **Parâmetros** | **Descrição** |
| --- | --- | --- | --- |
| NOP | 0 | - | Nenhuma operação |
| LOAD | 1 | Registrador, End. | Carrega valor da memória para registrador |
| CMP | 2 | Registrador, Reg. | Compara dois registradores |
| JMP | 3 | Endereço | Salta para um endereço |
| JEQ | 4 | Endereço | Salta se igual |
| JGR | 5 | Endereço | Salta se maior |
| STORE | 6 | Registrador, End. | Armazena valor do registrador na memória |
| MOV | 7 | Registrador, Reg. | Move valor entre registradores |
| ADD | 8 | Registrador, Reg. | Soma dois registradores |
| SUB | 9 | Registrador, Reg. | Subtrai dois registradores |
| AND | A | Registrador, Reg. | Operação AND entre dois registradores |
| OR | B | Registrador, Reg. | Operação OR entre dois registradores |
| NOT | C | Registrador | Operação NOT em um registrador |
| IN | D | Registrador | Carrega o valor das chaves no registrador |
| OUT | E | Registrador | Exibe o valor do registrador nos LEDs |
| WAIT | F | - | Aguarda o botão ser pressionado. |

## **Processo de Tradução**

1. **Primeira Passagem**
   * Processar rótulos e identificar endereços simbólicos.
   * Armazenar rótulos com seus endereços correspondentes.
2. **Geração de Código de Máquina**
   * Traduzir instruções e operandos para código hexadecimal.
   * Substituir rótulos por endereços temporários.
3. **Resolução de Endereços**
   * Ajustar endereços temporários após remoção de rótulos.
   * Substituir endereços temporários por endereços finais.