Pontifícia Universidade Católica do Rio Grande do Sul

FENG/FACIN

Arquitetura de Computadores I
Processador V2b

Nomes: Bruna Silva, Bruno Porcher, Vinicius Fochi
Turma: 590

Porto Alegre, 04 de Julho de 2008

SUMÁRIO

INTRODUÇÃO

O propósito desse trabalho prático, consiste na implementação da arquitetura de um processador de 8 bits em VHDL com suporte as principais operações realizadas por um processador com arquitetura pipeline, controle de hazard, operações aritméticas de adição e subtração, instruções de salto condicional ou imediato, carregamento e gravação de informações na memória de dados e memória de instruções.

1. CARACTERÍSTICAS GERAIS DA ARQUITETURA

O processador V2B possui um conjunto de instruções com 9 comandos, sendo que 8 bits são para a instrução e 8 bits para o endereço de memória, os registradores AC e PC possuem 8 bits cada e os registradores ID/EX e IF/ID possuem 16 bits cada.

A maior limitação da arquitetura V2b reside em seu barramento de endereços, que possui apenas 8 bits, o que determina um mapa de memória de apenas 256 posições disponíveis para armazenar programas e dados. O motivo de impor tal limitação é tão-somente facilitar o processo didático, pois durante o aprendizado de organização de computadores não é estritamente necessário escrever programas de grande porte ou lidar com grandes volumes de dados. As características restantes da arquitetura podem ser ainda hoje encontradas em processadores comerciais, tais como micro-controladores para uso em eletrônica embarcada de 8/16 bits da Intel (e.g. o 80C51) e Motorola (e.g. o 68HC11), entre outros.

Em termos de estrutura, trata-se de uma arquitetura muito simples, similar às introduzidas pela primeira vez na década de 1970, baseadas em um registrador de trabalho único, o Acumulador, fonte e destino da maioria das operações realizadas sobre dados.

O endereçamento de memória é orientado a byte, cada endereço de memória corresponde a uma posição onde residem apenas 8 bits. Uma palavra do processador, ao ser armazenada na memória de instruções ocupa apenas 2 bytes.

A arquitetura v2b possui um acumulador que armazena os resultados das operações, um registrador que determina a instrução a ser executada e

ainda possui dois registradores intermediários que armazenam as instruções do pipeline, são eles:

- Registrador PC (Program Counter), contém o endereço a ser lido na memória de instruções da próxima operação a ser executada. (8 bits do endereço)

- Registrador IF/ID, este registrador recebe a instrução de 16 bits diretamente do PC e faz um tratamento, decodificando-a e certificando-se de que não se trata de uma instrução de salto, assim a instrução poderá seguir adiante, pois as operações serão executadas nos ciclos posteriores. Caso seja identificada alguma instrução de salto dentro de IF/ID, cada caso será tratado de forma diferente, ou seja, caso a instrução seja “JMP loc”, o endereço de loc é passado imediatamente para o PC de modo que a próxima instrução a ser executada no pipeline já será a indicada pelo salto. Caso a operação a ser executada seja um BRN ou BRZ, o próprio IF/ID que é o responsável pelo incremento do PC, guarda o valor do PC e passa adiante duas instruções NOP, para que o processador possa finalizar alguma possível operação com o valor de PC.

- Registrador ID/EX, contém a instrução em uso na arquitetura em um determinado instante. Seu tamanho é de 16 bits, pois contém a instrução e o endereço de memória do operando. As instruções que são executadas após este registrador são: LDA, STA, ADD e SUB.
Ex.: A instrução ADD 00010000 será armazenada no ID/EX da seguinte maneira, a instrução ADD em binário será 00000010 então ID/EX terá a informação de 00000010 concatenado com 00010000 que é o endereço de memória do dado a ser somado com AC, portanto ID/EX = 0000001000010000.

- Registrador AC é um acumulador que armazena os resultados das operações e também carrega informações da memória de dados (8 bits).

2. CLASSES DE INSTRUÇÕES

Na arquitetura V2B podemos separar as nove operações, quanto a seu funcionamento, basicamente em 3 classes:

• Instruções de Movimento de Dados – Geralmente, é uma das classes mais utilizadas, dependendo do programa em questão, sobretudo em arquiteturas com poucos registradores, e inclui as instruções LDA e STA;

• Instruções Aritméticas – É a única classe cuja execução pode produzir transformações de dados sendo, portanto, fundamental em qualquer arquitetura. No V2B, a classe inclui as instruções lógicas ADD e SUB;

• Instruções de Controle do Fluxo de Execução – É a classe que permite comandar a ordem de execução das instruções restantes, sendo assim útil para acrescentar flexibilidade e tornar programas mais compactos. Inclui o desvio incondicional “JMP”, desvios condicionais “BRN e BRZ”, a instrução de parada HLT e NOP para espera de operações.

A arquitetura V2B foi projetada com base no conceito de que todas as operações de manipulação de dados e aritméticas usam de alguma forma o acumulador AC. Esta simplificação arquitetural permite que exista apenas um formato de instrução:

• Instruções com um operando explícito.

Instruções sem operando explícito ocupam apenas um byte, enquanto que instruções com um operando explícito ocupam dois bytes da memória de programa.

Tabela 1 – Conjunto de Instruções da Arquitetura V2B

Descrição da instrução	Instrução	Código	Descrição
Leitura da memória para o acumulador	LDA loc	00000000 loc	Carrega AC com o conteúdo da memória da posição especificada por loc. **AC <= Mem[loc]**
Escrita na memória	STA loc	00000001 loc	Armazena o valor de AC na posição de memória especificada por loc. **Mem[loc] <= AC**
Adição de valor ao valor do acumulador	ADD loc	00000010 loc	Adiciona o valor da memória definida por loc em AC ao valor de AC. **AC <= AC + Mem[loc]**
Subtração do valor do acumulador.	SUB loc	00000011 loc	Subtrai o valor da memória definida por loc em AC ao valor de AC. **AC <= AC - Mem[loc]**
Salto incondicional	JMP loc	00000100 loc	PC recebe valor imediato especificado pelo **operando**. **PC <= loc**
Salto condicionado ao valor do AC negativo	BRN loc	00000101 loc	Desvia para a instrução indicada pelo endereço loc. **Se (AC < 0) então PC <= loc**
Salto condicionado ao valor do AC = 0	BRZ loc	00000110 loc	Desvia para a instrução indicada pelo endereço loc. **Se (AC = 0) então PC <= loc**
Parada do Processador HLT	HALT	11111111 loc	Suspende a realização dos ciclos de busca, decodificação e execução.
Sem operação	NOP	00000111 loc	Não faz nada.

3. FLUXO DE EXECUÇÃO DAS INSTRUÇÕES

LDA

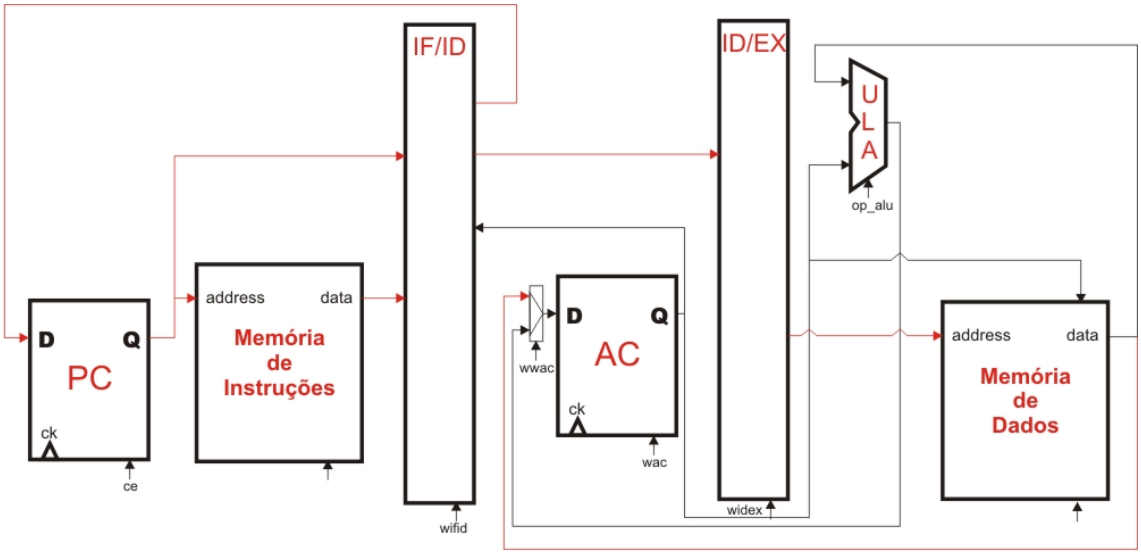

Figura 1

Ciclo 1 - PC passa o seu valor para a Memória de Instruções, que devolve a instrução e o dado para o IF/ID.
Ciclo 2 - IF/ID recebe e identifica a instrução na borda de subida e passa adiante os valores na borda de descida para o ID/EX.
Ciclo 3 - ID/EX passa o valor adiante. A Memória De Dados recebe o endereço vindo do ID/EX, e devido ao fato dela ser assíncrona devolve imediatamente o dado para o AC. O AC recebe o dado assincronamente estando pronto para utilizar a informação no ciclo seguinte.

STA

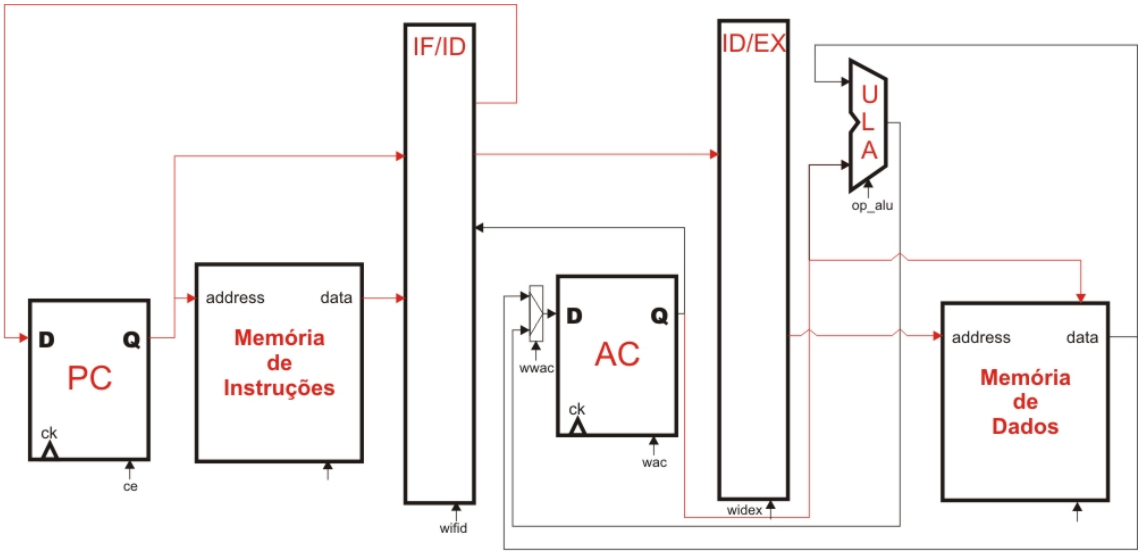

Figura 2

Ciclo 1- PC passa o seu valor para a Memória de Instruções, que devolve a instrução e o dado para o IF/ID.
Ciclo 2 - IF/ID recebe e identifica a instrução na borda de subida e passa adiante os valores na borda de descida para o ID/EX.
Ciclo 3 - ID/EX passa o valor adiante. A Memória De Dados recebe o endereço vindo do ID/EX, e o valor vindo do AC e devido ao fato dela ser assíncrona a gravação é imediata do dado vindo de AC.

ADD

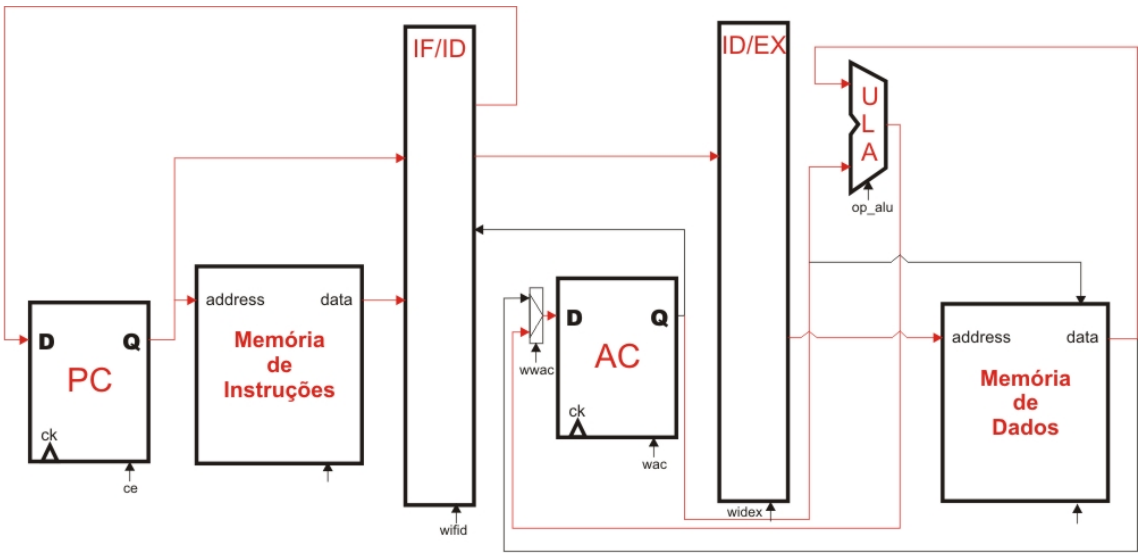

Figura 3

Ciclo 1 - PC passa o seu valor para a Memória de Instruções, que devolve a instrução e o dado para IF/ID.
Ciclo 2 - IF/ID recebe e identifica a instrução na borda de subida e passa adiante os valores na borda de descida para o ID/EX.
Ciclo 3 - ID/EX passa adiante o valor. A Memória de Dados recebe o endereço vindo do ID/EX, e repassa o valor do dado para a ULA na entrada op2 e devido ao fato dela ser assíncrona, no mesmo instante que a memória de dados está com o valor pronto o AC passa o valor para a ULA pela porta op1 coordenada pela unidade de controle. A ULA também é assíncrona ela soma os valores e já passa o valor da adição para o registrador AC, no mesmo ciclo. AC guarda o resultado.

SUB

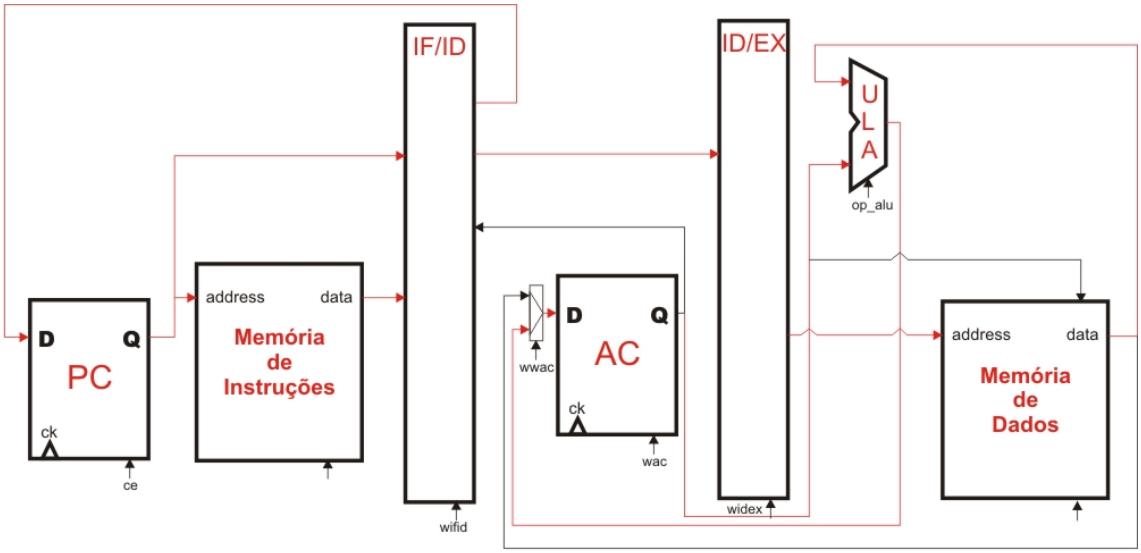

Figura 4

Ciclo 1 - PC passa o seu valor para a Memória de Instruções, que devolve a instrução e o dado para IF/ID.
Ciclo 2 - IF/ID recebe e identifica a instrução na borda de subida e passa adiante os valores na borda de descida para o ID/EX.
Ciclo 3 - ID/EX passa o valor adiante. A Memória de Dados recebe o endereço vindo do ID/EX, e passa o valor do dado para a ULA na entrada op2 e devido ao fato dela ser assíncrona, e no mesmo instante o AC passa o valor para a ULA pela entrada op1 coordenada pelo bloco de controle. A ULA também é assíncrona, portanto, ela calcula os valores e já passa o resultado da subtração para o registrador AC, no mesmo ciclo. AC guarda o valor.

JMP

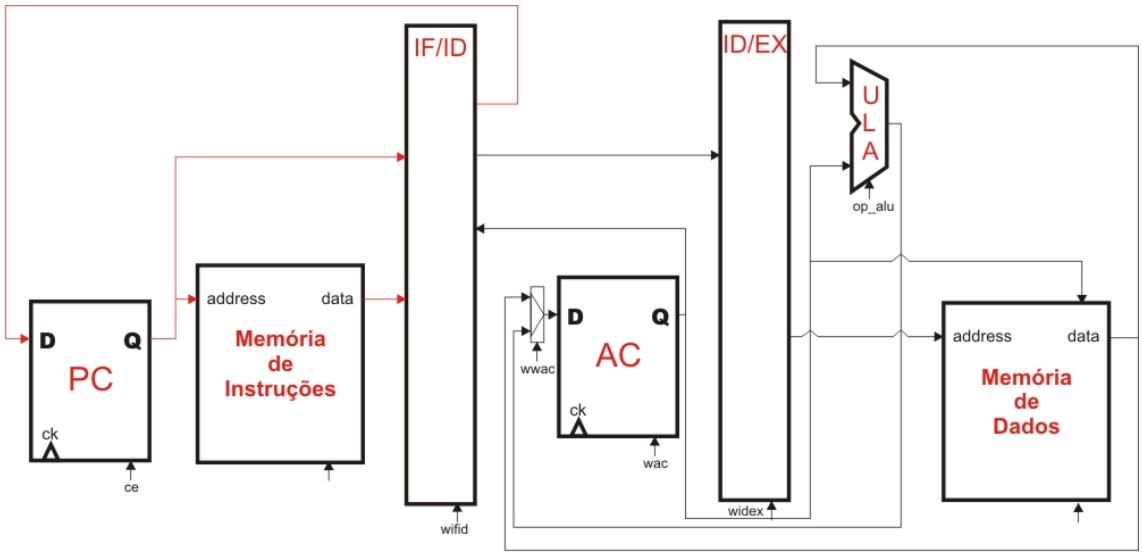

Figura 5

Ciclo 1 - PC passa o seu valor para a Memória de Instruções, que devolve a instrução e o dado para o IF/ID.
Ciclo 2 - IF/ID decodifica a instrução e já passa assincronamente para o PC o endereço da próxima instrução.

BRN

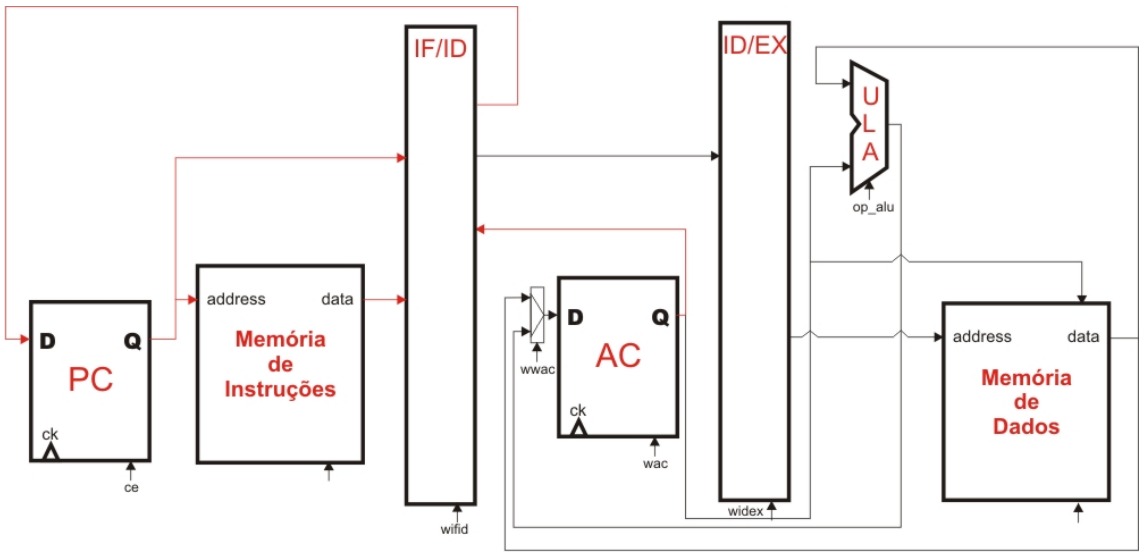

Figura 6

Ciclo 1 - PC passa o seu valor para a Memória de Instruções, que devolve a instrução e o dado para o IF/ID.
Ciclo 2 - IF/ID recebe e identifica a instrução de BRN e insere NOP nos próximos dois ciclos, após recebe o valor de AC e testa se AC < 0, se for então passa para o PC o endereço informado senão ele não faz nada e a próxima instrução é executada.
Ciclo 3 – NOP
Ciclo 4 – NOP

BRZ

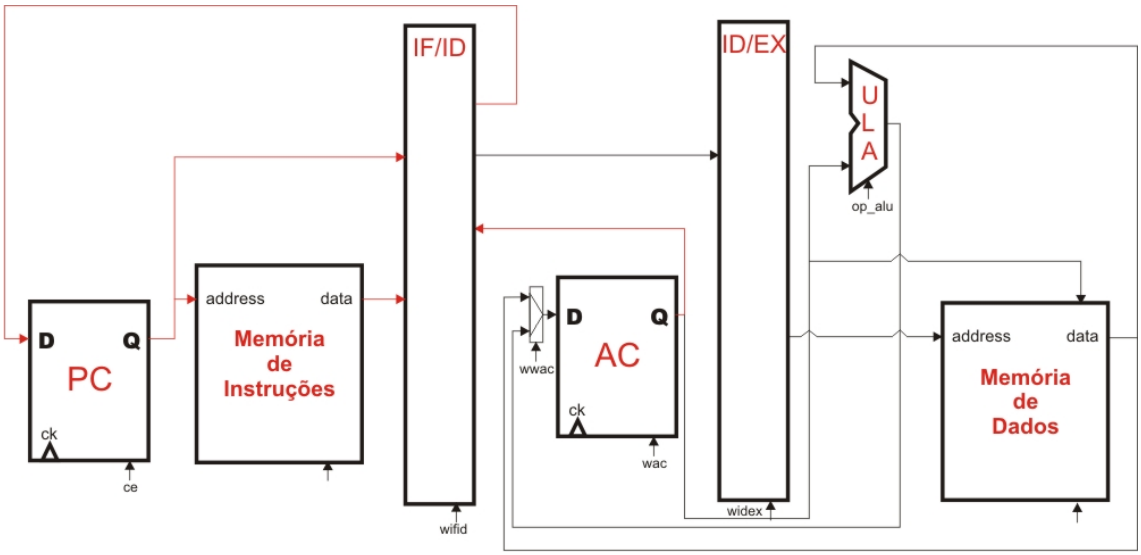

Figura 7

Ciclo 1 - PC passa o seu valor para a Memória de Instruções, que devolve a instrução e o dado para o IF/ID.
Ciclo 2 - IF/ID recebe e identifica a instrução de BRZ e insere NOP nos próximos dois ciclos, após recebe o valor de AC e testa se AC = 0, se for então passa para o PC o endereço informado senão ele não faz nada e a próxima instrução é executada.
Ciclo 3 – NOP
Ciclo 4 – NOP

4. A RELAÇÃO PROCESSADOR - MEMÓRIA – Mundo externo

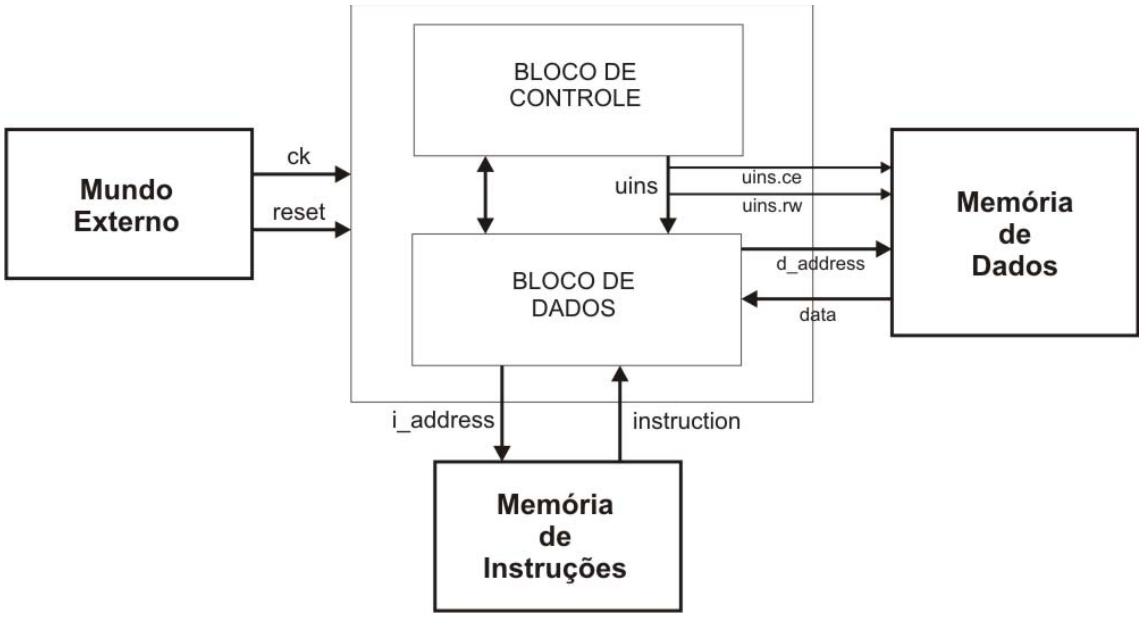

Figura 8

O mundo externo é o responsável por gerar os sinais de **clock** e **reset**.

O tamanho da memória de instruções e da memória de dados é de 256 bytes.

O **clock** sincroniza todos os eventos internos do processador. O sinal **reset** leva o processador a reiniciar a execução de instruções a partir do endereço (definir o endereço inicial) da memória.

Os sinais providos pelo processador MR4 para a troca de informações com as memórias são:

i_address – barramento de 8 bits que informa qual o endereço na memória de instruções que está a instrução a ser executada.

d_address – barramento unidirecional de 8 bits, contendo o endereço da posição de memória a ser acessada para leitura ou escrita de dados, da ou para a memória de dados, respectivamente;

data - barramento bidirecional de 8 bits transportando dados para o processador;

5. BLOCO DE CONTROLE

Sinais de entrada do Bloco de Controle:

ck - Clock do processador

rst - Sinal de reset

irp1 - std_logic_vector de 8 bits, representa qual instrução está no ifid no momento.

irp2 - std_logic_vector de 8 bits, representa qual instrução está no idex no momento

irp3 - std_logic_vector de 8 bits, representa qual instrução na saida do idex no momento

Sinais de saida do Bloco de Controle:

wac - Sinal de controle do Registrador AC – *wac <= "11" when i3=LDA or i3=ADDU or i3=SUBU else "01"*; -- habilita escrita no AC

wifid – Sinal de controle do IF/ID – quando wifid="00" o registrador IF/ID tem seus 16 bits em zero. Se wifid="11" então o registrador IF/ID passa adiante a instrução o dado e o valor de PC + 1.

widex – Sinal do ID/EX está sempre em fixo em "11" ele sempre passa o valor adiante para a memória de instruções não importando a instrução.

wwac - Sinal que determina a origem do valor que vai pro acumulador.
Se for LDA wwac<="11" ou seja o AC recebe o valor que vem da memória de dados, se for ADDU ou SUBU wwac<="01" o AC recebe o valor que vem da ULA.

wpc – Sinal de controle do PC está sempre em fixo em 1.

uins- A microinstrução uins é composta de três sinais:

uins.ce – Fica em 1 quando instrução for STA,LDA, ADD, SUB. Ficando em 1 ele libera a memória de dados para leitura e escrita, senão fica em '0'.

uins.rw – Fica em o quando a instrução for STA.

uins.i – Serve para decodificar a instrução. Exemplo: 0000000010100111 uins.i=LDA.

6. SIMULADOR DO PROCESSADOR

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>

int ram[5];
int AC;

struct mem_ins{
char comando[3];
int  end;   }; //endereco de memoria das intrucoes comeca em 10000000 igual a
ram

struct mem_ins mem[3];

//implementação do jmp
int jump(int x)
{
switch (x){
case 10000000:{
   return 0;
   break;  }

  case 10000001:{
  return 1;
  break;  }

  case 10000010:{
  return 2;
  break;  }

  case 10000011:{
  return 3;
  break;  }

  case 10000100:{
  return 4;
  break;  }

  case 10000101:{
  return 5;
  break;  }

default: printf("\nDigite um numero de memoria valido valido\n");
system("PAUSE"); break; }
}
```

```
//implementação do sub
void subu(int x)
{
switch (x){
case 10000000:{
   AC=AC-ram[0];
   break;  }

  case 10000001:{
  AC=AC-ram[1];
  break;  }

  case 10000010:{
  AC=AC-ram[2];
  break;  }

  case 10000011:{
  AC=AC-ram[3];
  break;  }

  case 10000100:{
  AC=AC-ram[4];
  break;  }
default: printf("\nDigite um numero de memoria valido valido\n");
system("PAUSE"); break; }
}

//implementação do add
void addu(int x)
{
switch (x){
case 10000000:{
   AC=AC+ram[0];
   break;  }

  case 10000001:{
  AC=AC+ram[1];
  break;  }

  case 10000010:{
  AC=AC+ram[2];
  break;  }

  case 10000011:{
  AC=AC+ram[3];
  break;  }
```

```
  case 10000100:{
  AC=AC+ram[4];
  break;  }
default: printf("\nDigite um numero de memoria valido valido\n");
system("PAUSE"); break; }
}

//implementação do STA
void store(int x)
{
switch (x){
  case 10000000:{
   ram[0]=AC;
   break;  }

  case 10000001:{
  ram[1]=AC;
  break;  }

  case 10000010:{
  ram[2]=AC;
  break;  }

  case 10000011:{
  ram[3]=AC;
  break;  }

  case 10000100:{
  ram[4]=AC;
  break;  }

default: printf("\nDigite um numero de memoria valido valido\n");
system("PAUSE"); break; }
}

//implementação do LDA
void load(int x)
{
switch (x){
  case 10000000:{
   AC=ram[0];
   break;  }

  case 10000001:{
  AC=ram[1];
  break;  }

  case 10000010:{
  AC=ram[2];
```

```
  break;  }

  case 10000011:{
  AC=ram[3];
  break;  }

  case 10000100:{
  AC=ram[4];
  break;  }

default: printf("\nDigite um numero de memoria valido valido\n");
system("PAUSE"); break; }
}

int main(int argc, char *argv[])
{
int f,x,u,jmp;
char ins[3];

ram[0]=0;
ram[1]=1;
ram[2]=2;
ram[3]=3;
ram[4]=4;
AC=0;

printf("Estado atual da memoria ram e AC\n");
printf("\n10000000 ram= %d",ram[0]);
printf("\n10000001 ram= %d",ram[1]);
printf("\n10000010 ram= %d",ram[2]);
printf("\n10000011 ram= %d",ram[3]);
printf("\n10000100 ram= %d\n",ram[4]);
printf("AC = %d",AC);
printf("\n\n");


printf("Digite o comando e o endereco de memoria:\n");
printf("obs: mem comecando em 10000000 e terminando em 10000100\n");

for(u=0;u<5;u++){
scanf("%s", mem[u].comando);
scanf("%d",&mem[u].end);
}

u=0;
do{

/*printf("Estado atual da memoria ram e AC\n");
printf("\n10000000 ram= %d",ram[0]);
printf("\n10000001 ram= %d",ram[1]);
```

```
printf("\n10000010 ram= %d",ram[2]);
printf("\n10000011 ram= %d",ram[3]);
printf("\n10000100 ram= %d\n",ram[4]);
printf("AC = %d",AC);
printf("\n\n");*/

strcpy(ins,mem[u].comando);
x=mem[u].end;

int p=-1;

p = strcmp(ins,"LDA"); if (p==0){ f=0;    };

p = strcmp(ins,"STA"); if (p==0){ f=1;    };

p = strcmp(ins,"ADD"); if (p==0){ f=2;    };

p = strcmp(ins,"SUB"); if (p==0){ f=3;    };

p = strcmp(ins,"JMP"); if (p==0){ f=4;    };

p = strcmp(ins,"BRN"); if (p==0){ f=5;    };

p = strcmp(ins,"BRZ"); if (p==0){ f=6;    };

p = strcmp(ins,"HAT"); if (p==0){ f=7;    };

//memoria de instruções
switch (f){
  case 0:{  load(x); break;  }
  case 1:{  store(x); break;  }
  case 2:{  addu(x); break; }
  case 3:{  subu(x); break; }
  case 4:{  jmp=jump(x); u=jmp-1; break;}
  case 5:{ if(AC<0){
           jmp=jump(x);
           u=jmp-1; }
            break;}

  case 6:{ if(AC==0){
           jmp=jump(x);
           u=jmp-1;}
            break; }
}

printf("\n");
printf("AC = %d",AC);
printf("\n10000000 ram= %d",ram[0]);
printf("\n10000001 ram= %d",ram[1]);
printf("\n10000010 ram= %d",ram[2]);
```

```
printf("\n10000011 ram= %d",ram[3]);
printf("\n10000100 ram= %d\n",ram[4]);
printf("\n");
system("PAUSE");

u++;
}while(u!=5);

  printf("\n");
  system("PAUSE");
  return 0;
  }
```

7. EXEMPLOS DE CODIGOS

Exemplos de um código:
ADD 10000100
SUB 10000001
LDA 10000000
STA 10000100
ADD 10000001

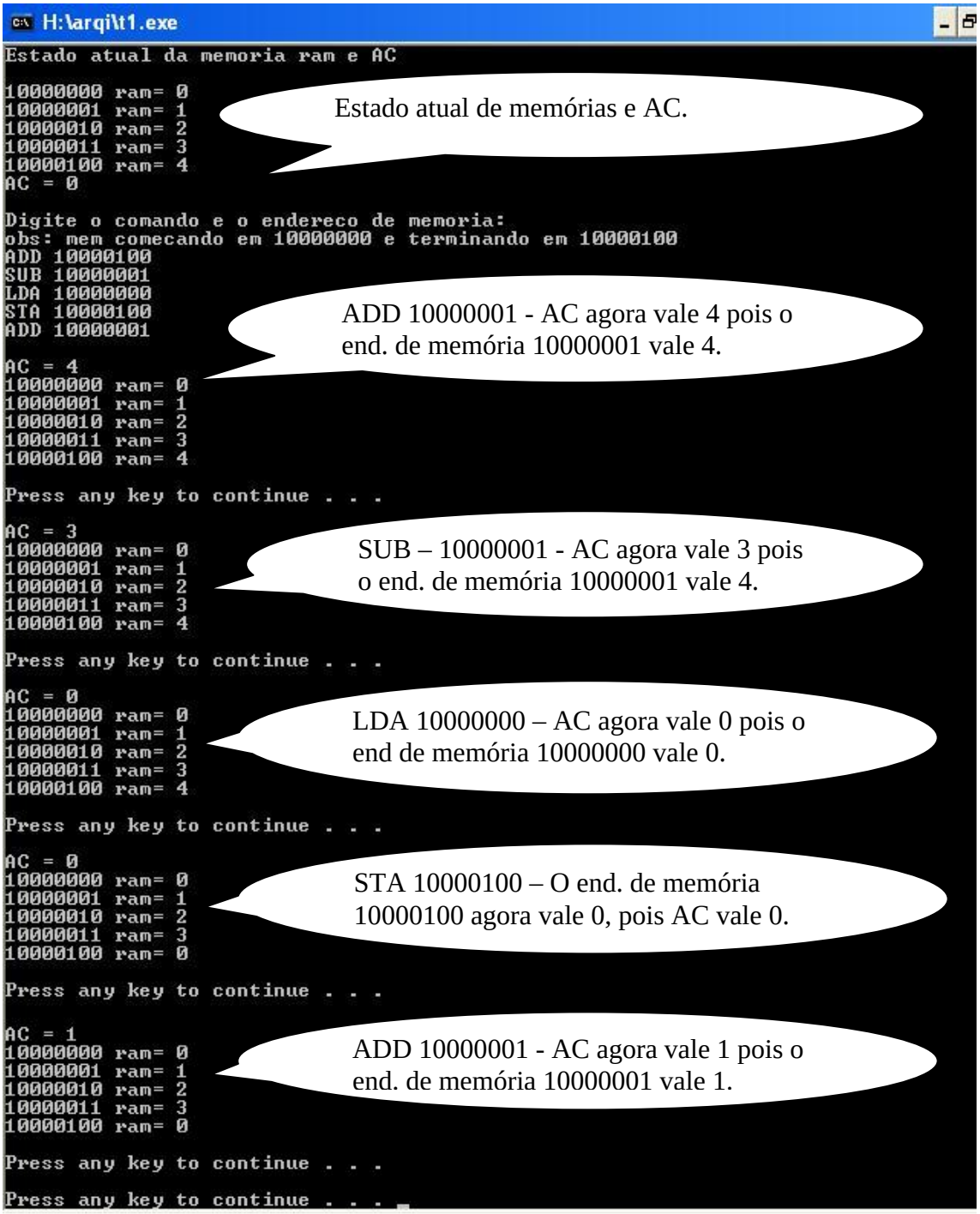

Figura 9

Exemplo do uso BRN:
10000000 ADD 10000011
10000001 SUB 10000100
10000010 BRN 10000100
10000011 ADD 10000001
10000100 ADD 10000100

Figura 10

Exemplo de código:
BRZ 10000100
SUB 10000100
ADD 10000100
SUB 10000100
SUB 10000100

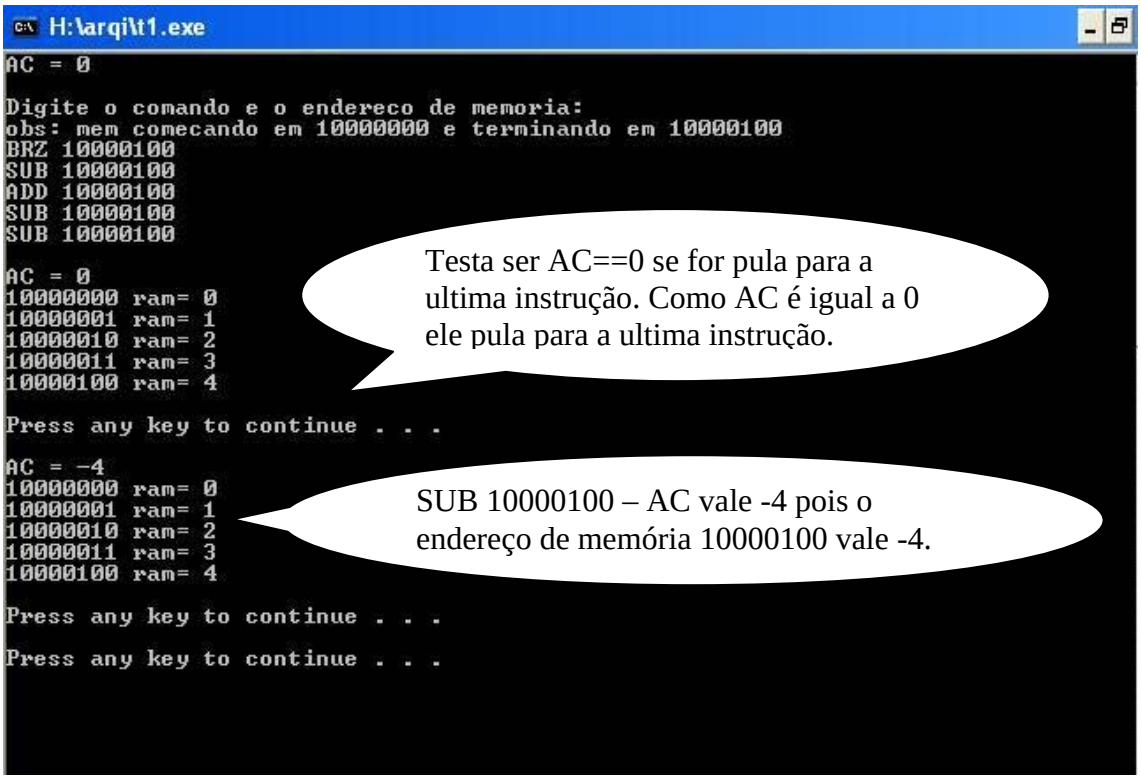

Figura 11

Exemplo de código:
JMP 10000100
ADD 10000001
ADD 10000011
ADD 10000100
SUB 100000001

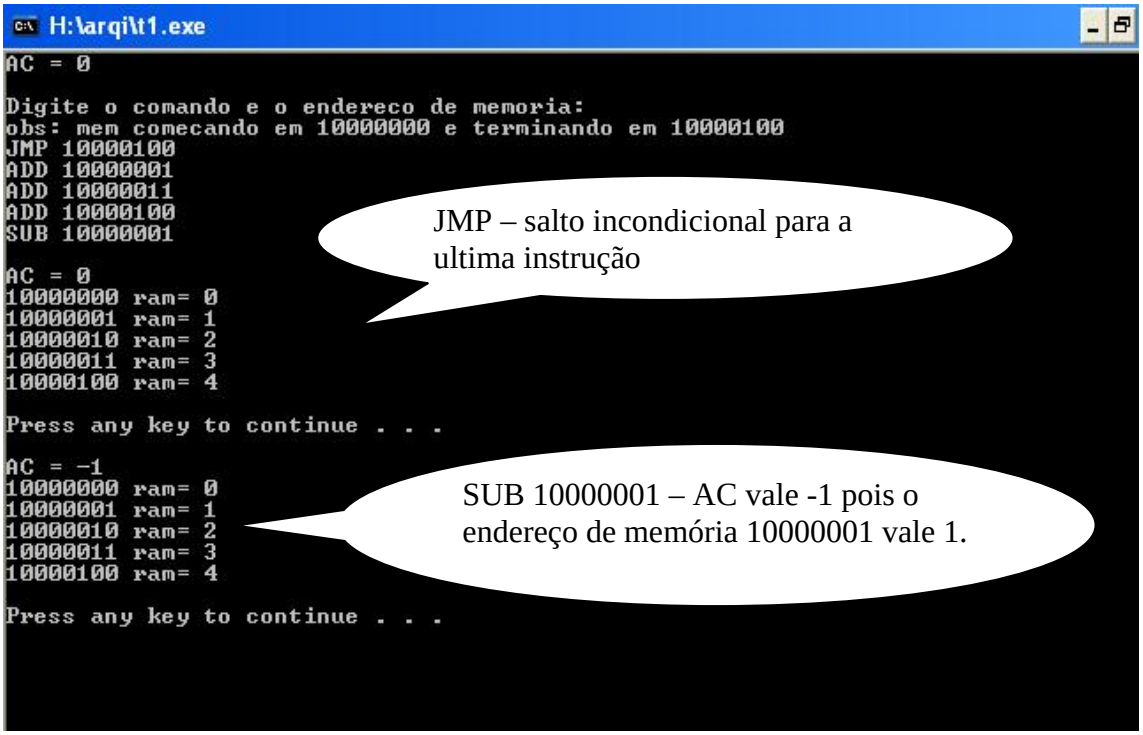

Figura 12

8. VISUALIZANDO O WAVEFORM (SIMULAÇÃO)