Processador: (Slide 35)

Unidade de busca -> Decodificação -> Executa

B -> D -> E -> B -> D -> E -> B -> D -> E -> B -> D -> E -> B -> D -> E -> … B -> D -> E -> ...

Formato de Instrução: nnnn Operando

Opcode (código operação)

PC (Program Counter): Endereço da próxima instrução

IR (Instruction Register): Instrução Corrente

AC (Cumulador): Memória Temporária

1 Instrução. 1940 Opcode= 1

Operando= 940

AC <- m[940]

2. 5941 Opcode= 5

Operando= 941

AC <- AC + m[941] //memória no acumulador

3. 2941 Opcode= 2

Operando= 941

m[941] <- AC

//Problemas: 1. Levaria muito tempo para fazer isso a mão, e o processador faz milhões de instruções em segundos (Por isso ele é o componente mais caro do computador). 2. Erros.

Memória:

PC:401

IR: 1403

AC: 9876

//1º. Buscar instrução no endereço 400

400: 1403

401: 2402

402: 1404

403: 2404

404: 1223

1403 -> AC <- m[403]

2402 -> m[402] <- AC

2404 -> m[404] <- AC

PC Original: 4,77 MHz

386: 16-40 MHz

Pentium: 100-166 MHz

Recentemente: 2,5-4 GHz

Paralelismo: Pipeline

Sem Pipeline:

B D E

1º i1

2º i1

3º i1

4º i2

5º i2

6º i2

7º i3

8º i3

9º i3

10º i4

11º i4

12º i4

…

Com Pipeline:

B D E

i1

i2 i1

i3 i2 i1

i4 i3 i2

i4 i3

i4

…

scanf(“%d”, &x);

if(x>10)

printf(“boo”);

else printf(“yah”);

Superescalar: uma evolução da pipeline.

Linguagem programação Assembly

Programador entra na aplicação ….

|  |
| --- |
| Aplicação |
| Bibliotecas |
| Runtime |
| SO |
| Hw |

Gerencia recursos

MU estendida

Como o processador trabalha com a instrução (implementa em assembler): x++;

MOV AX, (x)

INC AX //incrementar

MOV (x), AX //inverte

Outro exemplo: y= x-2 \* z;

MOV AX, (x)

MOV BX, (z)

SHL BX, 1 ; BX=BX<<1

SUB AX, BX

MOV (y), AX

Outro exemplo:

void f1 (void \*argp)

{

int i, temp;

for (i=0; i < MAX; i++){

temp=n;

temp++;

n=temp;

}

}

Não deu zero por causa de como o processador trabalha com a instrução.

n=0 [Global]

MAX=2000

Endereço Conteúdo

7ff8 ? <- SP

7ff6 0000

7ff4 0000

7ff2 0001

7ff0 0004 <- SP

7fee

Codificar em Assembler

Em C:

/\* acha p=log2(x) \*/

p=0;

n=1;

while (n<=x){

n\*=2;

p++;

}

p--;

Em Assembler:

Em vez de while é: desvio condicional, se determinada condição for verdadeira, desvia

x= CX, p=AX, n=BX

MOV AX, 0

MOV BX, 1

Início= CMP BX, CX

JA 2f; JA=Jump if above //desviar

SHL BX, 1 //deslocar

INC AX //incrementar

JMP Início

Fim: DEC AX //decrementar

Código em C para determinar se número é par ou ímpar

if(n%2==0)

printf (“par”);

else

printf (“ímpar”);