Instruções e saltos

ASC I 2015/2016

Teresa Gonçalves

Resumo

- Conjunto de instruções (*instruction set*)
- Branches condicionais e incondicionais.
- Implementação de IF, WHILE, FOR, etc

Execução de uma instrução

Consiste em:

1. Fetch da instrução apontada pelo PC; PC$\leftarrow$PC + 4.
2. Execução da instrução.

Ver exemplo no quadro de giz $\longrightarrow$

Algumas instruções aritméticas e lógicas (assembly)

`add $t0, $t1, $t2`	*# t0 = t1+t2*
`sub $t0, $t1, $t2`	*# t0 = t1-t2*
`sra $t0, $t1, 4`	*# t0 = shift right t1 by 4 bits (arith.)*
`addi $t0, $t1, 32767`	*# t0 = t1+32767*
`lui $t0, 0x1234`	*# t0 = 0x12340000*
`slt $t0, $t1, $t2`	*# t0 = 1 se t1<t2, 0 caso contrário*
`and $t0, $t1, $t2`	*# t0 = t1 and t2*
`or $t0, $t1, $t2`	*# t0 = t1 or t2*
`nor $t0, $t1, $t2`	*# t0 = not (t1 or t2)*
`xor $t0, $t1, $t2`	*# t0 = t1 xor t2*
`ori $t0, $t1, 0xf0f0`	*# t0 = t1 or 0x0000f0f0*
`andi $t0, $t1, 0xffff`	*# t0 = t1 and 0x0000ffff*
`sll $t0, $t1, 31`	*# t0 = shift left t1 by 31 bits*
`srl $t0, $t1, 4`	*# t0 = shift right t1 by 4 bits (logic)*

Branches - Saltos condicionais

As instruções de *branch* permitem mudar o rumo de execução de um programa se uma dada condição é satisfeita.

beq $t0, $t1, XPTO	*Branch to label XPTO if t0==t1*
bne $t0, $t1, ABC	*Branch to ABC if not equal*

Exemplo

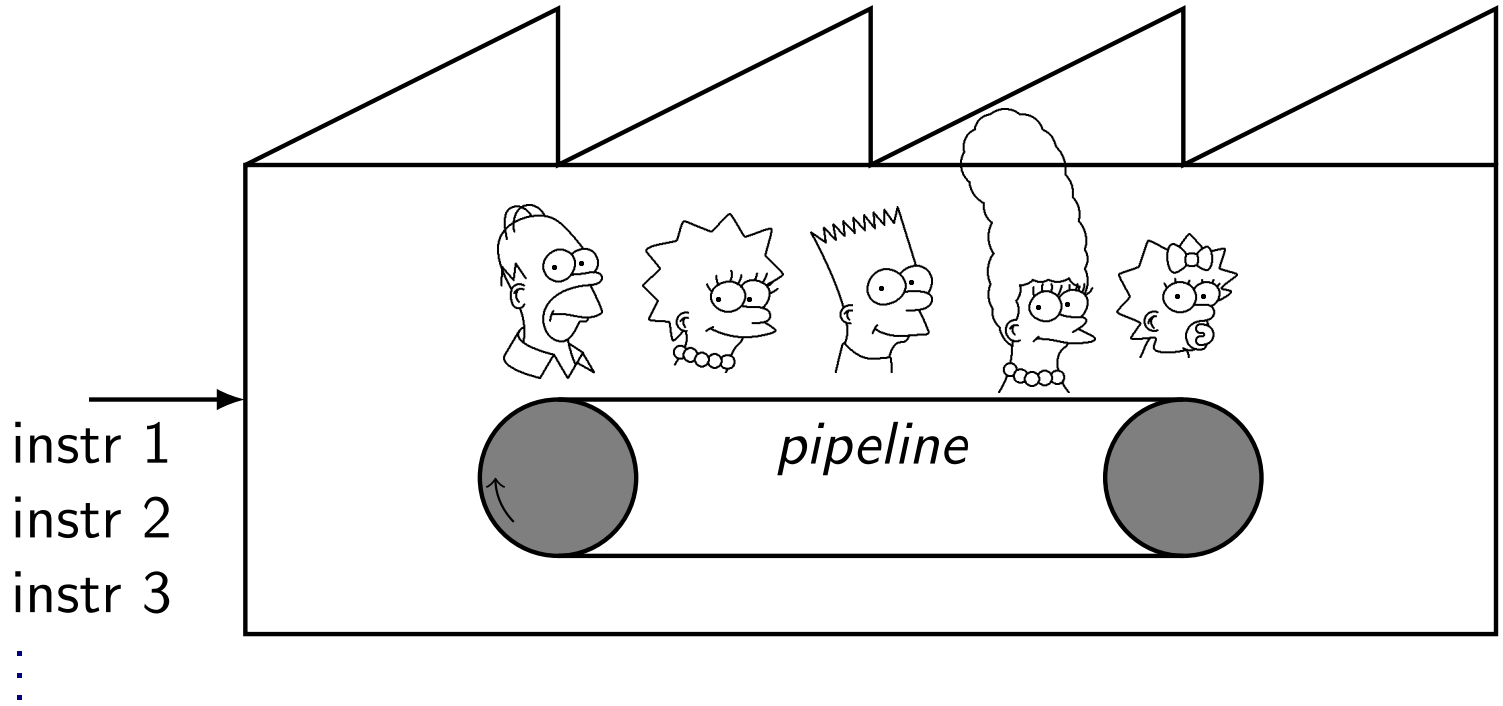
```
        addi $t0, $zero, 2
        addi $t1, $zero, 3
        bne  $t0, $t1, L1
        nop
        # o que estiver aqui não é executado

L1:
        # resto do programa...
```

Delayed Branches

Atenção

A instrução seguinte ao branch é **SEMPRE** executada, independentemente de o branch ser tomado ou não.



Quando a Lisa executa o branch, já o Homer foi buscar a instrução seguinte para o pipeline... D'oh!

Delayed Branches

Como a instrução no delay slot é sempre executada, temos duas opções:

Delayed Branches

Como a instrução no delay slot é sempre executada, temos duas opções:

1. Inserir um `nop` a seguir ao branch (nop=No OPeration). Exemplo:

```
        ...
addi $t2, $t2, 1
beq  $t0, $t1, L1
nop                    # delay slot (não faz nada)
addi $t1, $zero, 1
        ...
```

Delayed Branches

Como a instrução no delay slot é sempre executada, temos duas opções:

1. Inserir um `nop` a seguir ao branch (nop=No OPeration).
   Exemplo:

```
        ...
addi $t2, $t2, 1
beq  $t0, $t1, L1
nop                    # delay slot (não faz nada)
addi $t1, $zero, 1
        ...
```

2. Mover uma instrução que esteja antes do branch para o delay slot, desde que isso não altere o resultado final.
   Exemplo:

```
        ...
beq  $t0, $t1, L1
addi $t2, $t2, 1     # esta estava antes do beq
addi $t1, $zero, 1
        ...
```

Implementação de IF/THEN/ELSE

Em linguagem C/C++:

```
if (t0 == t1)
{
  //
  // A
  //
}
else
{
  //
  // B
  //
}
```

Em assembly:

```
        bne $t0, $t1, ELSE
        nop
        #
        # A
        #
        beq $zero, $zero, END
        nop

ELSE:
        #
        # B
        #

END:
```

Implementação do ciclo FOR

Em linguagem C/C++:

```
for (t0=0; t0<t1; t0++)
{
  // enquanto t0 < t1
  // executa instrucoes
  // neste espaco.
  //
  // incrementa t0:
  //       t0 = t0+1
  //
  // e repete.
}
```

Em assembly:

```
        add $t0, $zero, $zero

FOR:    slt $t2, $t0, $t1
        beq $t2, $zero, ENDFOR
        nop
        #
        # executa instrucoes
        # neste espaco.
        #
        addi $t0, $t0, 1
        beq $zero, $zero, FOR
        nop
ENDFOR:
```

Implementação do ciclo DO/WHILE

Em linguagem C/C++:

```
do
{
  //
  // executa instrucoes
  // neste espaco.
  //
  // repete se t0 >= t1
  //
} while (t0 >= t1)
```

Em assembly:

```
DO:
        #
        # executa instrucoes
        # neste espaco.
        #

        slt $t2, $t0, $t1
        beq $t2, $zero, DO
        nop
```

Exercício: Implementação do SWITCH/CASE

Em linguagem C/C++:

```
switch (t0)
{
  case 1:
          // a
  case 2:
          // b
          break;

  case 3:
          // c
  default:
          // d
}
```

Em assembly:

```
# Como se faz?
#
# Veja como funciona o switch
# em linguagem C.
#
# Implemente essa funcionalidade
# em assembly.
```