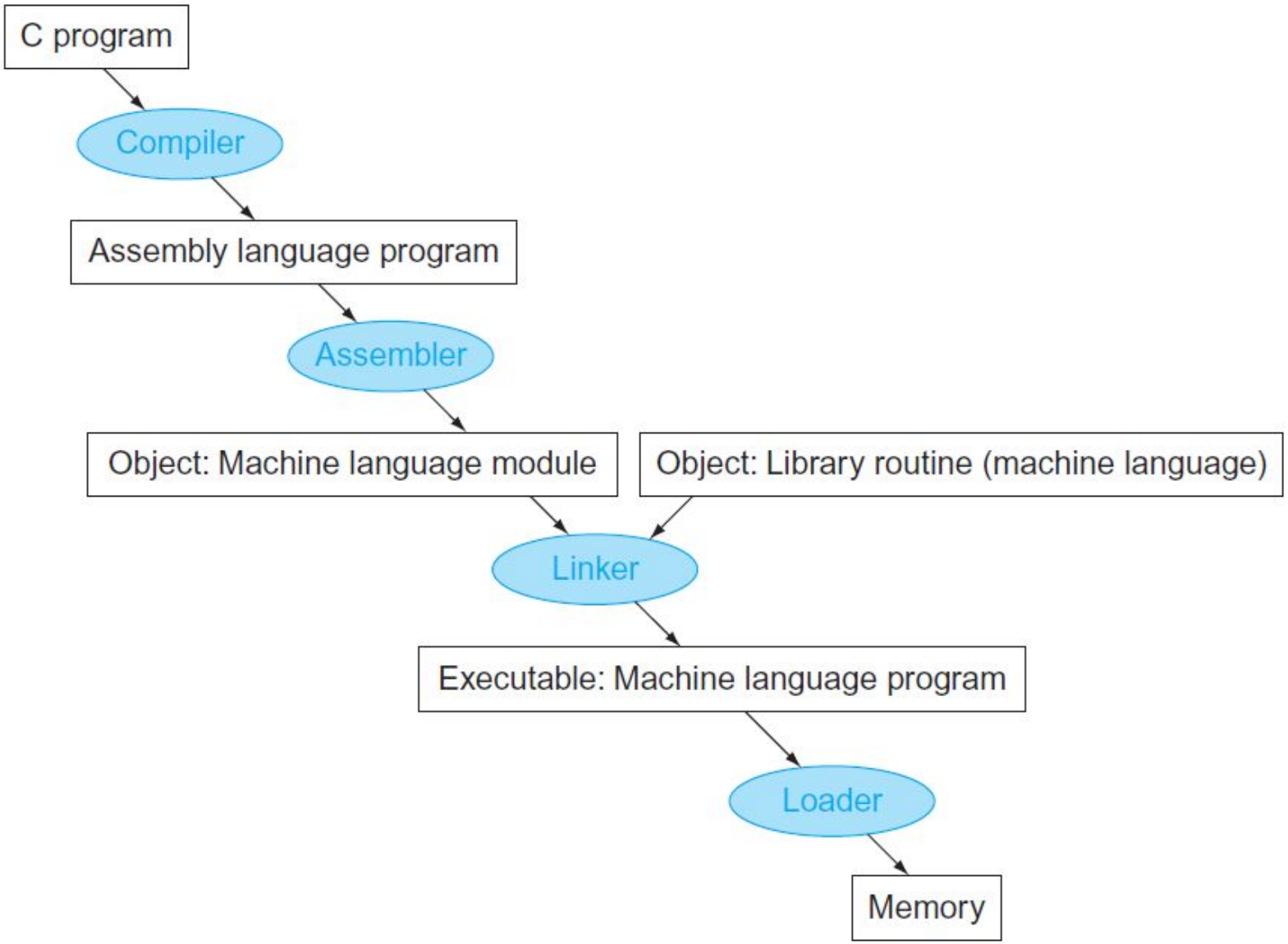
Aula 3

Conjunto de Instruções

Conjunto de Instruções

- É a “linguagem” do hardware
- Semelhante a uma “linguagem de programação restrita”: poucos comandos
- Projetistas de processadores tem um objetivo em comum: criar uma linguagem de hardware que seja de fácil implementação e tradução para compiladores de outras linguagens


C program
Compiler
Assembly language program
Assembler
Object: Machine language module
Object: Library routine (machine language)
Linker
Executable: Machine language program
Loader
Memory

Bubblesort no gcc

gcc optimization	Relative performance	Clock cycles (millions)	Instruction count (millions)	CPI
None	1.00	158,615	114,938	1.38
O1 (medium)	2.37	66,990	37,470	1.79
O2 (full)	2.38	66,521	39,993	1.66
O3 (procedure integration)	2.41	65,747	44,993	1.46

Bubblesort vs Quicksort

Language	Execution method	Optimization	Bubble Sort relative performance	Quicksort relative performance	Speedup Quicksort vs. Bubble Sort
C	Compiler	None	1.00	1.00	2468
	Compiler	01	2.37	1.50	1562
	Compiler	02	2.38	1.50	1555
	Compiler	03	2.41	1.91	1955

Conjunto de Instruções

- O conjunto de instruções deve ser escolhido de maneira a simplificar o projeto dos circuitos internos do processador.
- Conhecer o conjunto de instruções permite compreender o funcionamento do processador e as decisões de melhoria de desempenho
- O conjunto de instruções estudados na disciplina são dos processadores ARM e MIPS

Conjunto de Instruções

- O conjunto básico de instruções de um processador são as operações aritméticas
  - add a,b,c : soma duas variáveis b,c e armazena em a.
  - a=(b+c)+(d+e)
    - add a, b,c
    - add a,a,d
    - add a,a,e

Conjunto de Instruções

- Em um conjunto de instruções aritméticas em um processador como o ARM/MIPS, as operações sempre executam sobre 3 operandos
  - *Simplicidade favorece regularidade*
- Regularidade implica em um projeto de processador mais simples

Conjunto de Instruções

- Relação de uma linguagem de alto nível com MIPS
- O código em C

```
a = b + c;
d = a - e;
```

- Compilado para MIPS

```
add a, b, c
sub d, a, e
```

Conjunto de Instruções

- Código em C

```
f = (g + h) - (i + j);
```

- Equivalente em MIPS:
  - O compilador cria variáveis temporarias (t0,t1) para quebrar a operação:
    - add t0,g,h # faz t0=(g+h)
    - add t1,i,j # faz t1=(i+j)
    - sub f,t0,t1# faz f=(t0-t1)

Conjunto de Instruções

- Diferente das linguagens de programação de alto nível, as linguagens de montagem possuem uma quantidade restrita de variáveis
- Estas variáveis são pequenos espaços de memória dentro do processador chamados de *Registradores*
- Os registradores são organizados em regiões continuas no processador chamadas de *Banco de Registradores*

Conjunto de Instruções

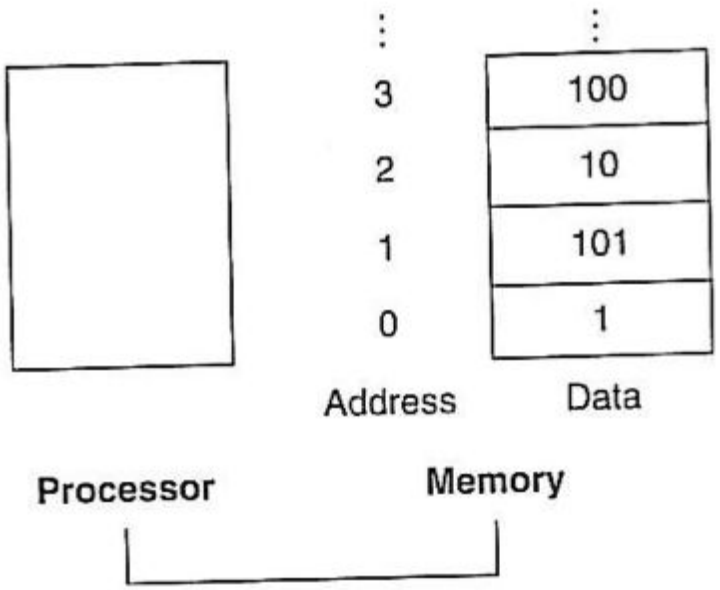
- Na arquitetura MIPS, um registrador tem o tamanho de 32 bits – tamanho também conhecido como “palavra”
- Nesta arquitetura, o banco de registradores armazena no máximo 32 registradores. Significa que durante a compilação, o compilador pode utilizar no máximo 32 “variáveis”
- A nomenclatura dos registradores do MIPS utiliza $ no inicio: $s1, $s1, $t0, $t1

Conjunto de Instruções

- Operandos da memória
  - As variáveis dos programas são armazenadas na memória principal
  - Para serem executadas na arquitetura MIPS, as variáveis precisam ser carregadas nos registradores
  - Instruções de transferência de dados

Conjunto de Instruções

- Acesso a memória
  - Load → carrega memória no registrador
  - Store → carrega registrador na memória



Conjunto de Instruções

- Código em C
  - a=b+c;
- Equivalente MIPS

```
lw $s1,b
lw $s2,c
add $s3,$s1,$s2
sw $s3,a
```

Conjunto de Instruções

- O conjunto básico de instruções de um processador são as operações aritméticas
  - add a,b,c : soma duas variáveis b,c e armazena em a.
  - a=(b+c)+(d+e)
    - add a, b,c
    - add a,a,d
    - add a,a,e

Conjunto de Instruções

- Considere A, um vetor int A[100]. O compilador C associou s registradores $s1 e $s2 para as variáveis g e h, respectivamente. Dado o código:
  - g=h+A[8];

Qual o equivalente em MIPS?

Conjunto de Instruções

- O código faz uma operação direto na memória, então, primeiro este valor deve ser transferido para um registrador.
- O vetor na posição A[8] i é representado pelo valor base A, mais o deslocamento 8. A[8] <-> *(A+8)
- Considerando que $s3 contém a base A temos
  - lw $s3,A
  - lw $t0,8($s3) # $t0 recebe o valor A[8] -> 8($s3)
- Por fim
  - add $s1,$s2,$t0

Conjunto de Instruções

- No MIPS, devido o tamanho dos registradores serem de 32 bits, o endereço de memória também usa 32 bits, 4 bytes. Por isso, cada acesso a memória tem o ajuste de 4.
- Dado o código
  - A[12]=h+A[8]

Se char A[20]

seu equivalente:

```
lw $t0, 8($s3)  # $t0 recebe A[8]
add $t0,$s2,$t0 # $t0 recebe h+A[8]
sw $t0,12($s3) # A[12] recebe $t0
```

Se int A[20]

seu equivalente:

```
lw $t0, 32($s3)  # $t0 recebe A[8]
add $t0,$s2,$t0 # $t0 recebe h+A[8]
sw $t0,48($s3) # A[12] recebe $t0
```

Se float A[20] #64

seu equivalente:

```
lw $t0, 64($s3)  # $t0 recebe A[8]
add $t0,$s2,$t0 # $t0 recebe h+A[8]
sw $t0,96($s3) # A[12] recebe $t0
```

Conjunto de Instruções

- Devido ao acesso a memória, instruções lw e st possuem um CPI maior, limitado pelo tempo de acesso ao dado na memória
- No caso de operadores constantes, o MIPS tem suporte a instruções de operação direta
  - i=i+4; → addi $s1,$s1,4
  - São operações comuns e são as mais rápidas em uma arquitetura
    - *O caso comum deve ser o mais rápido*

Exercicios

Para cada exercicio, procure usar a menor quantidade possível de registradores

1) f= g + (h+ 5);
2) f=f+f+i;
3) f=f+g+h+i+j+2;
4) f=g+h+B[4]
5) f=g-A[B[4]];

1) f= g + (h+ 5);

R:

```
lw $s0,h
lw $s1,g
addi $s0,$s0,5
add $s0,$s0,$s1
sw $s0,f
```

2) f=f+f+i;

```
lw $s0,f
lw $s2,f
lw $s1,I
add $s0,$s0,$s1 #f+i
add $s0,$s2,$s0 #f+(f+i)
sw $s0,f
```

3) f=f+g+h+i+j+2;

```
lw $s0,j
addi $s1,$s0,2
lw $s0,i
add $s1,$s1,$s0
lw $s0,h
add $s1,$s1,$s0
lw $s0,g
add $s1,$s1,$s0
lw $s0,f
add $s1,$s1,$s0
sw $s1,f
```

4) f=g+h+B[4]

```
lw $s0,B
lw $t0, 4($s0)
lw $s1,h
lw $s2,g
add $s3,$s1,$t0
add $s3, $s3,$s2
sw $s3,f
```

5) f=g-A[B[4]]

```
lw $s0, B
lw $t0, 4($s0)
lw $s1, A
add $t1, $s1,$t0 # $t1 -> (*A+B[4])
lw $s2, 0($t1)
lw $s3,g
sub $s3,$s3,$s2
sw $s3,f
```

Instruções de salto incondicional

- Em alguns casos, é necessário mudar o fluxo do programa:
  - Instrução J L1 → muda o fluxo do programa para a linha L1
  - while(1) i++; ->

L1:

addi $s1,$s1,i

J L1

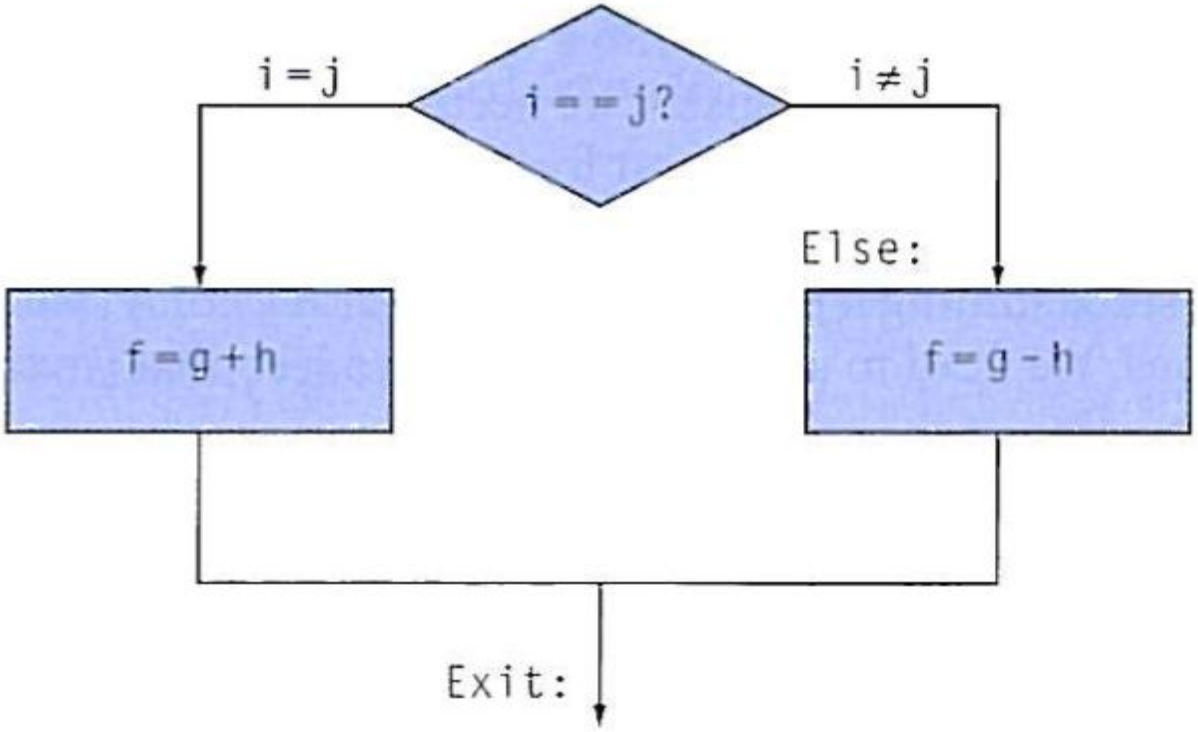
Instruções de tomada de decisão

- A tomada de decisões afeta o fluxo do programa, de acordo com alguma verificação lógica
- Instrução para verificar se dois registradores tem o mesmo valor
  - beq registrador1, registrador2, L1
    - beq → branch if equal. Caso registrador1 for igual registrador2, o fluxo deve continuar na linha L1
  -

- bne registrador1, registrador2, L1
  - bne → branch if NOT equal. Caso registrador1 for diferente registrador2, o fluxo deve continuar na linha L1

Instruções de tomada de decisão

- Código em C
  - If(i==j) f=g+h; else f=g-h;



Instruções de tomada de decisão

- Considerando que as variáveis de f a g estão associadas aos registradores $s0 a $s4:

```
    bne $s3,$s4,Else # Va para Else se i!=j
    add $s0,$s1,$s2 # f = g + h
    J Exit
Else: sub $s0,$s1,$s2
Exit:
```

Instruções de tomada de decisão

- Considerando que as variáveis de f a g estão associadas aos registradores $s0 a $s4:

```
    bne $s3,$s4,Else # Va para Else se i!=j
    add $s0,$s1,$s2 # f = g + h
    J Exit
Else: sub $s0,$s1,$s2
Exit:
```

Instruções de tomada de decisão

- Considerando que as variáveis de f a g estão associadas aos registradores $s0 a $s4:

```
        bne $s3,$s4,Else # Va para Else se i!=j
        add $s0,$s1,$s2 # f = g + h
        J Exit
Else: sub $s0,$s1,$s2
Exit:
```

Instruções de tomada de decisão

- Considerando que as variáveis de f a g estão associadas aos registradores $s0 a $s4:

```
    bne $s3,$s4,Else # Va para Else se i!=j
    add $s0,$s1,$s2 # f = g + h
    J Exit
Else: sub $s0,$s1,$s2 # f=g-h
Exit:
```

Instruções de tomada de decisão

- Codigo em C
  - while(save[i]==k)

i+=1;

- MIPS
  - i,k são salvos em $s3 e $s5. A base do vetor save está salva em $s6
  - Instrução sll → shift a direita. Equivale a multiplicar por 2 a cada shitf. Sll $s1,$s1, 1 → i=i*2;

Instruções de Tomada de decisão

- while(save[i]==k)

  i++;

- MIPS

```
Loop:sll $t1,$s3,2      # Temp reg $t1 = 4 * i
     add  $t1,$t1,$s6   # $t1 = address of save[i]
     lw   $t0,0($t1)    # Temp reg $t0 = save[i]
     bne  $t0,$s5, Exit # go to Exit if save[i] ≠ k
     addi $s3,$s3,1     # i = i + 1
     j    Loop          # go to Loop
Exit:
```

Instruções de Tomada de decisão

- while(save[i]==k)

  i++;

- MIPS

```
Loop: sll $t1,$s3,2       #$t1 = i*4
      add $t1,$t1,$s6 # $t1 = endereço de save[i]
      lw $t0,0($t1)    #$t0=save[i]
      bne $t0,$s5,Exit # se save[i]!=k, sai
      addi $s3,$s3,1   #i++;
      J Loop             # vai pra Loop
```

Instruções de tomada de decisão

- As instruções de tomada de decisão possuem um papel fundamental em uma linguagem de programação
- No projeto de compiladores, trechos SEM instruções de tomada de decisão são chamados de blocos básicos
  - Por afetarem o fluxo do programa, a tomada de decisão também afeta o desempenho final

Instruções de tomada de decisão

- Outras instruções do MIPS
  - slt,slti→ set on less than.
    - slt $t0, $s1,$s2 → se ($s1 <$s2) $t0 =1 , senao $t0=0
    - slti $t0, $s1,5 → se ($s1 <5) $t0 =1 , senao $t0=0
- if(i<4) →

  slti $t0,$s0,4

  bne $t0,$zero,L1

```
void strcpy (char x[], char y[])
{
    int i;

    i = 0;
    while ((x[i] = y[i]) != '\0') /* copy & test byte */
    i += 1;
}
```

```
strcpy:
    addi   $sp,$sp,-4   # adjust stack for 1 more item
    sw     $s0, 0($sp)  # save $s0
    add    $s0,$zero,$zero # i = 0 + 0

L1: add    $t1,$s0,$a1  # address of y[i] in $t1

    lbu    $t2, 0($t1)  # $t2 = y[i]
    add    $t3,$s0,$a0  # address of x[i] in $t3
    sb     $t2, 0($t3)  # x[i] = y[i]
    beq    $t2,$zero,L2 # if y[i] == 0, go to L2
    addi   $s0, $s0,1   # i = i + 1
    j      L1           # go to L1

L2: lw     $s0, 0($sp)  # y[i] == 0: end of string.
                        # Restore old $s0
    addi   $sp,$sp,4    # pop 1 word off stack
    jr     $ra          # return
```

```
while (save[i] == k)
       i += 1;
```

```
Loop:sll $t1,$s3,2      # Temp reg $t1 = 4 * i
     add  $t1,$t1,$s6   # $t1 = address of save[i]
     lw   $t0,0($t1)    # Temp reg $t0 = save[i]
     bne  $t0,$s5, Exit # go to Exit if save[i] ≠ k
     addi $s3,$s3,1     # i = i + 1
     j    Loop          # go to Loop
Exit:
```

Exercicio

Considere que as variáveis f,g,h,i, j foram usadas e sao associadas aos registradores $S0 ao $S4 respectivamente. $S6 e $S7 sao os vetores A e B. Faça o equivalente em C ao codigo assembler a seguir

```
sll  $t0, $s0, 2     # $t0 = f * 4
add  $t0, $s6, $t0   # $t0 = &A[f]
sll  $t1, $s1, 2     # $t1 = g * 4
add  $t1, $s7, $t1   # $t1 = &B[g]
lw   $s0, 0($t0)     # f = A[f]
addi $t2, $t0, 4
lw   $t0, 0($t2)
add  $t0, $t0, $s0
sw   $t0, 0($t1)
```