Instruções RISC-V

Rodolfo Azevedo

MC404 - Organização Básica de Computadores e Linguagem de Montagem

http://www.ic.unicamp.br/~rodolfo/mc404

Contexto

- Nos slides anteriores, vimos algumas das instruções e registradors do RISC-V
- Agora vamos avançar ampliando o conjunto de instruções e também trazendo novos registradores

Meta da aula de hoje

Ser capaz de entender e implementar em assembly códigos que contenham as seguintes estruturas:

- if
- if-else
- while
- for

if then else

> Se uma dada condição for verdadeira (if), então execute um trecho de código. Caso contrário (else), execute outro trecho de código.

```
if (x == 5)
  a += 7;
else
  a += 15;
```

while

> Enquanto uma dada condição for verdadeira, execute um trecho de código.

```
x = 20;
y = 10;
while (x != y)
{
  x += 2;
  y += 3;
}
```

for

> Execute um trecho de código um número fixo de vezes.

```
a = 0;
for (i = 0; i < 100; i ++)
  a += i;
```

Relembrando os Registradores

Registrador	Descrição
zero	Valor fixo em zero (0)
ra	Endereço de retorno de chamada de função
sp, gp, tp	Apontador de pilha, dados globais e de thread
t0-t6	Valores temporários
s0-s11	Valores salvos
a0-a7	Argumentos para função e valores de retorno
pc	Contador de programa

Instruções de comparação

Instrução	Formato	Uso
Set less than	R	SLT rd, rs1, rs2
Set less than immediate	I	SLTI rd, rs1, imm
Set less than unsigned	R	SLTU rd, rs1, rs2
Set less than unsigned immediate	I	SLTIU rd, rs1, imm

Exemplo

- Como saber se i < j?

```
slt t0, t1, t2   # onde t1 deve ter o valor de i, t2 de j e t0 terá o resultado
```

- Se i < 0
  - t0 = 1
- Caso contrário
  - t0 = 0

Instruções de salto

> São instruções que desviam a execução do programa para um endereço diferente da próxima instrução

Instrução	Formato	Uso
Jump and link	J	JAL rd, label
Jump and link register	J	JALR rd, rs1, imm

Instruções de saltos condicionais

Instrução	Formato	Uso
Branch if ==	B	BEQ rs1, rs2, label
Branch if !=	B	BNE rs1, rs2, label
Branch if <	B	BLT rs1, rs2, label
Branch if >=	B	BGE rs1, rs2, label
Branch if < unsigned	B	BLTU rs1, rs2, label
Branch if >= unsigned	B	BGEU rs1, rs2, label

⚠️ Você pode inverter a ordem dos operandos se necessário!

Exemplo

- Se x == 0, some z = y + 5, caso contrário z = y + 7

```
# supondo que t0 tenha o valor de x, t1 de y e t2 de z
    beq t0, zero, e_zero
    addi t2, t1, 7
    j fim
e_zero:
    addi t2, t1, 5
fim:
```

Pseudo-instruções

> *Pseudo-instruções não são instruções reais, mas são úteis para escrever código mais legível. O montador converteem uma ou mais instruções reais*

Instrução	Descrição	Conversão
li a0, constante	Carrega uma constante em um registrador	Utiliza lui + addi para compor a constante
la a0, label	Carrega o endereço de uma label em um registrador	Utiliza auipc, mv e ld se necessário
mv a0, a1	Move o valor de um registrador para outro	Utiliza addi

Outros exemplos de pseudo-instruções

Instrução	Descrição	Conversão
call label	Chama uma função	Utiliza jal ou jalr
nop	Nenhuma operação	Utiliza addi zero, zero, 0
j destino	Salta para um destino	Utiliza jal zero, 0
ret	Retorna de uma função	Utiliza jalr zero, ra 0
not rd, rs	Inverte os bits de um registrador	xori rd, rs, -1

Exemplos de código em alto nível vs Assembly

Para cada código em C, faça uma implementação em assembly para o RISC-V

if then else

```
if (x == 5)
  a += 7;
else:
  a += 15;
```

Resolução

```
main:
  addi t1, zero, 9
  add t2, zero, zero
  addi t0, zero, 5
  bne t1, t0, else

  addi t2, t2, 7
  j fim
else:
  addi t2, t2, 15

fim:
  jr    ra
```

while

```
x = 20;
y = 10;
while x != y
{
  x += 2;
  y += 3;
}
```

Resolução

```
main:
  addi t0, zero, 20    # x
  addi t1, zero, 10    # y

loop :
  beq t0, t1, fim      # x == y ? vá para o fim
  addi t0, t0, 2
  addi t1, t1, 3
  j loop

fim:
  jr ra                # retorne
```

for

```
a = 0;
for (i = 0; i < 100; i ++)
  a += i;
```

Resolução

```
main:
  addi t0, zero, 0     # i
  addi t1, zero, 0     # a
  addi t2, zero, 100   # constante 100 para limite do for

for:
  bge t0, t2, fim      # i >= 100 ? vá para o fim
  add t1, t1, t0       # a += i
  addi t0, t0, 1       # i++
  j loop               # vá para o início do loop

fim:
  jr ra                # retorne
```