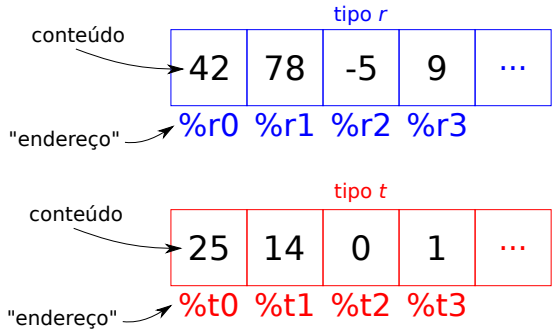
Compiladores

prof. Ricardo Oliveira

Documentação do *Assembly* Raposeitor

Raposeitor

O *Raposeitor* é um processador simplificado que contém dois conjuntos de registradores (células de memória): os registradores tipo r (`%r0, %r1, %r2, ...`) e os registradores tipo t (`%t0, %t1, %t2, ...`):



Não existe nenhuma diferença teórica entre os dois tipos de registradores, mas os registradores tipo r são usados para alocar as variáveis declaradas no programa (como *r*egistradores em si), enquanto os registradores tipo t são usados para armazenar valores intermediários durante o cálculo de expressões aritméticas e lógicas (como registradores *t*emporários).

Assim, nenhuma variável deve ser alocada nos registradores tipo t, e nenhum valor intermediário deve ser calculado em registradores tipo r.

Seu *assembly* tem um conjunto de instruções simplificado representado por códigos de três endereços, descritos a seguir:

Instruções de Entrada/Saida

- `printf str` imprime a *frase* `str`. A string deve obrigatoriamente estar entre aspas duplas (`""`). Para imprimir aspas, escape com contra-barra (`"\""`). Use `"\n"` para imprimir uma quebra de linha. Esta instrução imprime apenas strings.

  Exemplo:

  ```
  printf "Hello World!\n"
  ```

  Imprime `Hello World!` seguido de uma quebra de linha.

- `printv op` imprime o *valor* do operando `op`. O operando pode ser um inteiro ou um registrador.

  Exemplo:

  ```
  printv 42
  ```

  Imprime `42`.

  ```
  printv %r1
  ```

  Imprime o conteúdo do registrador `%r1` (no exemplo dado na figura acima, imprime 78).

- `read reg` lê um valor inteiro do usuário e o atrubui no registrador `reg`.

  Exemplo:

  ```
  read %r0
  ```

  Lê um valor do usuário e o atribui no registrador `%r0` (no exemplo, se o usuário entrar com o valor 42, o conteúdo do registrador passa a ser 42).

Instrução de Atribuição

- `mov rdest, op` atribui o operando `op` ao registrador de destino `rdest`. O operando pode ser um valor inteiro ou um registrador.

  Exemplo:

  ```
  mov %t1, 14
  ```

  atribui o valor 14 ao registrador `%t1`.

  ```
  mov %t4, %r3
  ```

  atribui o valor do registrador `%r3` (no exemplo, 9) ao registrador `%t4`.

Instruções de Operações Aritméticas

- `add rdest, op1, op2` soma os operandos `op1` e `op2` e atribui o resultado ao registrador de destino `rdest` (isto é, `rdest` recebe `op1 + op2`). Tanto `op1` quanto `op2` podem ser valores inteiros ou registradores. O registrador de destino pode ser o mesmo de algum operando, ou não.

  Exemplo:

  ```
  add %t5, 2, 8
  ```

  Atribui $2 + 8 = 10$ ao registrador `%t5`.

  ```
  add %t4, %r1, 32
  ```

  Atribui a soma do conteúdo do registrador `%r1` com 32 ao registrador `%t4` (no exemplo, `%t4` recebe 78+32=110).

  ```
  add %t6, %r2, %r3
  ```

  Atribui a soma dos conteúdos de `%r2` com `%r3` ao registrador `%t6` (no exemplo, `%t6` recebe -5+9=4).

- `sub rdest, op1, op2` é análogo à instrução anterior, mas com a operação de subtração (isto é, `rdest` recebe `op1 - op2`).

- `mult rdest, op1, op2` é análogo à instrução anterior, mas com a operação de multiplicação (isto é, `rdest` recebe `op1 * op2`).

- `div rdest, op1, op2` é análogo à instrução anterior, mas com a operação de quociente de divisão inteira (isto é, `rdest` recebe `op1 / op2`).

- `mod rdest, op1, op2` é análogo à instrução anterior, mas com a operação de resto de divisão inteira (isto é, `rdest` recebe `op1 % op2`).

Instruções de Operações Lógicas

- `not rdest, op` atribui ao registrador de destino `rdest` o valor 0 se o operando `op` for diferente de 0, ou o valor 1 se o operando `op` for igual a 0 ("negação"). O operando pode ser um valor ou um registrador.

  Exemplo:

  ```
  not %t7, 1
  ```

  atribui o valor 0 ao registrador `%t7`.

```
not %t7, %t2
```

atribui a %t7 o valor 1 se %t2 for 0, ou o valor 0 caso contrário (no exemplo, como %t2=0, atribui o valor 1).

- `or rdest, op1, op2` atribui ao registrador de destino `rdest` o valor 1 se algum dos operandos forem diferentes de 0, ou o valor 0 caso contrário ("ou"). Ambos os registradores podem ser valores inteiros ou registradores.

  Exemplo:

  ```
  or %t14, %t2, %t3
  ```

  atribui em %t14 o valor 1 se %t2 ou %t3 for diferente de 0, ou o valor 0 caso contrário (no exemplo, como %t3=1, atribui o valor 1).

- `and rdest, op1, op2` atribui ao registrador de destino `rdest` o valor 1 se ambos os operandos forem diferentes de 0, ou o valor 0 caso contrário ("e"). Ambos os registradores podem ser valores inteiros ou registradores.

  Exemplo:

  ```
  and %t14, %t2, %t3
  ```

  atribui em %t14 o valor 1 se ambos %t2 e %t3 forem diferentes de 0, ou o valor 0 caso contrário (no exemplo, como %t2=0, atribui o valor 0).

Instruções de Comparação

- `equal rdest, op1, op2` atribui ao registrador de destino `rdest` o valor 1 se o operando `op1` for igual ao operando `op2` (isto se, se `op1` = `op2`), ou 0 caso contrário. Tanto o operando `op1` quanto o operando `op2` podem ser valores inteiros ou registradores.

  Exemplo:

  ```
  equal %t6, %r0, 42
  ```

  atribui ao registrador %t6 o valor 1 se o conteúdo do registrador %r0 for igual a 42, ou 0 caso contrário (no exemplo acima, como %r0 = 42, atribui o valor 1).

- `diff rdest, op1, op2` é análogo à instrução anterior, mas comparando se `op1` $\neq$ `op2`.

- `less rdest, op1, op2` é análogo à instrução anterior, mas comparando se `op1` $<$ `op2`.

- `lesseq rdest, op1, op2` é análogo à instrução anterior, mas comparando se `op1` $\leq$ `op2`.

- `greater rdest, op1, op2` é análogo à instrução anterior, mas comparando se `op1` $>$ `op2`.

- `greatereq rdest, op1, op2` é análogo à instrução anterior, mas comparando se `op1` $\geq$ `op2`.

Instruções de Acesso Indireto

- `load rdest, desl(base)` atribui no registrador de destino `rdest` o conteúdo do registrador tipo r %rE, onde E é calculado pela soma de `base` e `desl` (isto é, `rdest` recebe `%r(base+desl)`).

  Tanto `base` quanto `desl` podem ser valores inteiros ou registradores.

  Exemplo:

  ```
  load %t4, 2(%t3)
  ```

  atribui ao registrador %t4 o conteúdo do registrador %rE, sendo E o conteúdo de %t3 mais 2 (no exemplo, como %t3 + 2 = 1 + 2 = 3, atribui o conteudo de %r3=9).

- `store orig, desl(base)` atribui o valor de origem `orig` ao registrador tipo *r* `%r`E, onde E é calculado pela soma de `base` e `desl` (isto é, `%r(base+desl)` recebe `orig`). Todos `orig`, `base` e `desl` podem ser valores inteiros ou registradores.

  Exemplo:

  ```
  store %r0, 5(%r3)
  ```

  atribui o conteúdo do registrador `%r0` ao registrador `%r`E, sendo E o conteúdo de `%r3` mais 5 (no exemplo, como `%r3` + 5 = 9 + 5 = 14, e como `%r0` = 42, atribui o valor 42 no registrador `%r14`).

  ```
  store 28, 5(%r3)
  ```

  atribui o valor 28 ao registrador `%r`E, sendo E o conteúdo de `%r3` mais 5 (no exemplo, como `%r3` + 5 = 9 + 5 = 14, atribui o valor 28 no registrador `%r14`).

Instruções de fluxo

- `label rotulo` rotula o local onde aparece no programa com o rótulo dado. Não executa nenhuma operação de fato.

  Exemplo:

  ```
  label R00
  ```

  indica que o local onde esta "instrução" aparece é o ponto de rótulo `R00` do programa.

- `jump rotulo` faz com que a execução do programa vá para o local indicado pelo rótulo dado ("*go-to* incondicional").

  Exemplo:

  ```
  jump R00
  ```

  altera o local de execução do programa para o indicado pelo rótulo `R00`.

- `jf op, rotulo` faz a execução ir para o local indicado pelo rótulo caso o operando `op` seja igual a 0 ("*jump if false*"). O operando pode ser um valor inteiro ou um registrador.

  Exemplo:

  ```
  jf %t2, R00
  ```

  altera o local de execução do programa para o indicado pelo rótulo `R00` caso o valor do registrador `%t2` seja igual a 0 (no exemplo dado, é o caso).

- `jt op, rotulo` é análogo à instrução anterior, mas com o local de execução alterado caso o operando seja diferente de 0 ("*jump if true*").

Comentários

O *Raposeitor* admite comentários de linha, com o caracter `;`.

Como exemplo, o seguinte código em *assembly* do *raposeitor* lê um valor do usuário e indica se ele é par ou impar:

```
; Le um numero e indica se eh par ou impar

printf "Digite um numero: "
read %r0                    ; Le N

printf "O numero "
printv %r0
printf " eh "

mod %t0, %r0, 2
equal %t1, %t0, 0           ; if (N % 2 == 0)
jf %t1, Relse

printf "par.\n"             ; imprime "par"
jump Rfim

label Relse                 ; else

printf "impar.\n"           ; imprime "impar"

label Rfim                  ; fim do if
```

Execução

Você pode rodar os códigos em *assembly* do Raposeitor gerados pelo seu compilador usando um interpretador. São fornecidos dois interpretadores: um em C++ e outro em Python3. Você pode usar o que preferir.

Executar com o interpretador em C++

1. Salve o código *assembly* em um arquivo (por exemplo, `programa.rap`);
2. Compile o programa `raposeitor.cpp`;
3. O execute passando o código por linha de comando:
   `$ ./raposeitor programa.rap`

Executar com o interpretador em Python3

1. Salve o código *assembly* em um arquivo (por exemplo, `programa.rap`);
2. Execute o Python3 passando o interpretador e o código por linha de comando:
   `$ python3 raposeitor.py programa.rap`