System Calls

- Umożliwia realizację operacji we/wyj (biblioteka „we/wyj”) w symulatorze SPIM
- Rejestr $v0 jest używany jako rejestr sterujący – wskazuje na operację do wykonania (tabela)
- Argumenty sa ładowane do rejestrów $a0 oraz $a1.
- Odpowiedź w rejestrze $v0 (operacje czytania)
- Sugerowana kolejność instrukcji
  - Załaduj rejestry argumentu
  - Wprowadź kod „operacji”
  - Wykonaj „syscall”

```
li  $a0, 10     # load argument $a0=10
li  $v0, 1      # call code to print integer
syscall         # print $a0
```

System Calls

Service	System Call Code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		

2

Dyrektywy Asemblera - wybrane

.text - sekcja kodu, po tej dyrektywie występują instrukcje asemblera

.data - sekcja danych

.globl - specyfikacja etykiet globalnych (adres symboliczny)

Przykład 1

```
.text                    # sekcja kodu
    .globl main
main:   li $v0, 4        # ustawienie drukowania „stringu”
        la $a0, str      # adres „stringu do wydruku”
        syscall          # drukuj „string”
        li $v0, 10       # ustawienie wyjścia
        syscall          # wyjście
    .data                # sekcja danych
str:  .asciiz “Hello world!\n”   # NUL kończy string (jak w C)
```

Przykłady we/wyj

```
.data        #  sekcja danych
    promptInt: .asciiz „Wprowadz wartosc typu „int” "
    resultInt: .asciiz „Nastepna wartosc = "
    linefeed: .asciiz "\n"
    enterkey: .asciiz „Nacisnij dowolny klawisz aby zakonczyc"
.text        # sekcja kodu
  main:
# zaproszenie do wprowadzenia wartosci „int”
      li $v0,4                  # kod dla wydruku „napisu”
      la $a0,promptInt          # w $a0 adres napisu
      syscall                   # wydrukuj zaproszenie
# pobranie „int” z wejścia
      li $v0,5                  # kod odczytu „int”
      syscall                   # pobierz „int” – wynik w $v0
      move $t0,$v0              # „int” w $t0
# wylicz następną wartość „int”
      addi $t0, $t0, 1          # t0 <-- t0 + 1
```


5

Przykłady we/wyj

```
# wydrukuj kolejna wartosc
        li $v0,4                    # kod dla wydruku „napisu”
        la $a0,resultInt            # w $a0 adres napisu
        syscall                     # wydrukuj „napis”
# print out the result
       li $v0,1                     # kod wydruku „int”
       move $a0,$t0                 # wynik w  $a0
       syscall                      # wydrukuj wynik
# przejscie do nowej linii
       li $v0,4                     # kod dla wydruku „napisu”
       la $a0,linefeed              # w $a0 adres napisu
       syscall                      # przejscie do nowej lini
# oczekiwanie na nacisniecie klawisza
        li $v0,4                    # kod dla wydruku „napisu”
        la $a0,enterkey             # w $a0 adres napisu
        syscall                     # wydrukuj „napis”
# oczekiwanie na nacisniecie klawisza – odczytana wartosc jest ignorowana
        li $v0,5                    # kod odczytu „int”
        syscall                     # pobierz „int” – wynik w $v0
# Koniec
        li $v0,10                   # kod wyjscia
        syscall                     # wyjscie z programu
```


6

Load i Store

- Dwie dodatkowe instrukcje komunikujące się z pamięcią **lb** oraz **sb** – operują na bajtach
- Taki sam format jak **lw** i **sw**
  - **lb $s2, 3($s1)**
  - **sb $s2, 3($s1)**
- Wykorzystywany jest najmniej znaczący bajt
- Bit znaku bajtu jest kopiowany na pozostałe bity słowa
- Aby temu zapobiec można zastosować instrukcję **lbu** – ładuj bajt bez znaku

Nadmiar w instrukcjach arytmetycznych

Dwa typy instrukcji arytmetycznych

Zgłaszające nadmiar

- add (**add**)
- add immediate (**addi**)
- subtract (**sub**)

Nie zgłaszające nadmiaru

- add unsigned (**addu**)
- add immediate unsigned (**addiu**)
- subtract unsigned (**subu**)

Operacje logiczne

Przesunięcia (shift)

sll $t2, $s0, 2 # t2 = s0 << 2 - w lewo

- **przed 0000 0002hex**
- **po : 0000 0008hex**

Format instrukcji

000000	00000	10000	01010	00010	000000
op	rs	rt	rd	shamt	funct

srl – przesunięcie w prawo

9

Operacje logiczne

Wykonywane na ciągu bitów

- Dwie standardowe instrukcje **and** oraz **or**
  - and $t0, $t1, $t2
  - or $t0, $t1, $t2
- Instrukcja **nor** zamiast **not**
- **Dlaczego ?**
- **A nor 0 = not (A or 0) = not A**
- Dodatkowo instrukcje **andi** oraz **ori** na operandach bezpośrednich

10

Instrukcje iteracji

- Prosta pętla w C; A[] – tablica typu int

```
do  { g = g + A[i];
      i = i + j;
    } while (i != h);
```

- Zamieniamy na następującą:

```
Loop: g = g + A[i];
      i = i + j;
      if (i != h) goto Loop;
```

- Zakładamy następujące przyporządkowanie

g,	h,	i,	j,	base of A
$s1,	$s2,	$s3,	$s4,	$s5

Instrukcje iteracji

```
Loop:  sll $t1,$s3,2          # $t1= 4*I
       addu $t1,$t1,$s5       # $t1=addr A+4i
       lw $t1,0($t1)          # $t1=A[i]
       addu $s1,$s1,$t1       # g=g+A[i]
       addu $s3,$s3,$s4       # i=i+j
       bne $s3,$s2,Loop       # goto Loop
                              # if i!=h
```

```
Loop: g = g + A[i];
      i = i + j;
      if (i != h) goto Loop;
```

g, h, i, j, base of A

$s1, $s2, $s3, $s4, $s5

12

Instrukcje iteracji

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

- Zakładamy następujące przyporządkowanie

i,	k,	base of save
$s3,	$s5,	$s6

```
loop:  sll $t1, $s3, 2
       add $t1, $t1, $s6
       lw $t0, 0($t1)
       bne $t0, $s5, exit
       addi $s3, $s3,1
       j  loop
exit:
```


13

Przykład 2 –we/wyj 1/2

```
.text
    .globl main
main:   la $t0, Aaddr       # $t0 = wskaznik na tablicę A
        lw $t1, len         # $t1 = rozmiar tablicy A
        sll $t1, $t1, 2     # $t1 = 4*rozmiar
        add $t1, $t1, $t0   # $t1 = za ostatnim elementem A
```


14

Przykład 2 –we/wyj 2/2

```
Loop:   lw   $t2, ($t0)          # $t2 = A[i]
        addi $t2, $t2, 5         # $t2 = $t2 + 5
        sw   $t2, ($t0)          # A[i] = $t2
        addi $t0, $t0, 4         # i = i+1
        bne  $t0, $t1, loop      # if $t0<$t1 goto loop
        li $v0, 10               # exit
       syscall
.data
   Aaddr:          .word 0,2,1,4,5
   len:    .word 5
```


15

Porównania

- Dotąd sprawdzaliśmy tylko dwa warunki (== oraz !=)
- Potrzebne są również inne, w szczególności „<„ oraz „>”
- Instrukcja **“Set on Less Than”**
  - **slt reg1,reg2,reg3**
- **Znaczenie** reg1 = (reg2 < reg3)

```
if (reg2 < reg3)
    reg1 = 1;
else reg1 = 0;
```

- Również instrukcja **slti**
- Zauważmy, że brak jest instrukcji **„less than”**

16

Porównania

- Jak wykorzystać tę instrukcję ?

if (g < h) goto Less; # g:$s0, h:$s1

- Odpowiedź:

```
slt $t0,$s0,$s1      # $t0 = 1 if g<h
bne $t0,$0,Less      # goto Less
                     # if $t0!=0
                     # (if (g<h)) Less:
```

- W **$0 zawsze wpisana wartość 0, stąd bne oraz beq są często uzywane wraz z slt**
- Para (**slt, bne) oznacza if(… < …)goto…**

Porównania

- Wiemy jak zaimplementować $<$,
- Ale jak zaimplementować $>$, $\leq$ oraz $\geq$?
- Może trzeba dodać trzy kolejne instrukcje
- Ale jak jest łatwiej ?
- Czy możemy zaimplementować $\leq$ stosując slt i skoki warunkowe ?
- A co zrobić z $>$? oraz $\geq$?

18

Porównania

```
                              # a:$s0, b:$s1
slt   $t0,$s0,$s1             # $t0 = 1 if a<b
beq $t0,$0,skip               # skip if a >= b
                              # do if a<b
```

skip:

- Inny wariant jest również możliwy
- Użyj **slt $t0,$s1,$s0** zamiast **slt $t0,$s0,$s1**, oraz **bne** zamiast **beq**

19

Porównania

- Jest również wersja **slt** z operandem bezpośrednim **slti**
- Użyteczna do tworzenia pętli
- if (g >= 1) goto Loop

```
Loop: . . .
slti $t0,$s0,1        # $t0 = 1 if
                      # $s0<1 (g<1)
beq $t0,$0,Loop       # goto Loop
                      # if $t0==0
                      # (if (g>=1))
```

- Para (slt, beq) oznacza if(… ≥ …)goto…

20

Porównania

- **Są również instrukcje porównania dla liczb bez znaku sltu, sltiu**
- **Ustawiają one wynik na 1 lub 0 w zależności od porównania wartości bez znaku**
- Dla $t0 oraz $t1 równych

($s0 = FFFF FFFA hex, $s1 = 0000 FFFA hex), jaki otrzymujemy wynik następujących porównań ?

```
slt $t0, $s0, $s1
sltu $t1, $s0, $s1
```


21

Switch

- **Wybierz jedną z czterech możliwości w zależności od wartości k 0, 1, 2 lub 3.**
- **Kod w C**

```
switch (k) {
case 0: f=i+j; break;   /* k=0 */
case 1: f=g+h; break; /* k=1 */
case 2: f=g–h; break; /* k=2 */
case 3: f=i–j; break;   /* k=3 */
}
```


22

Switch

- **Jest to dość złożone, przekształćmy więc kod tak aby użyć if else**

```
if(k==0) f=i+j;
else if(k==1) f=g+h;
else if(k==2) f=g–h;
else if(k==3) f=i–j;
```

Zakładamy następujące przyporządkowanie

f:$s0, g:$s1, h:$s2,

i:$s3, j:$s4, k:$s5

23

Switch

```
bne $s5,$0,L1        # branch k!=0
add $s0,$s3,$s4      #k==0 so f=i+j
j Exit               # end of case so Exit
L1: addi $t0,$s5,-1  # $t0=k-1
bne $t0,$0,L2        # branch k!=1
add $s0,$s1,$s2      #k==1 so f=g+h
j Exit               # end of case so Exit
L2: addi $t0,$s5,-2  # $t0=k-2
bne $t0,$0,L3        # branch k!=2
sub $s0,$s1,$s2      #k==2 so f=g-h
j Exit               # end of case so Exit
L3: addi $t0,$s5,-3  # $t0=k-3
bne $t0,$0,Exit      # branch k!=3
sub $s0,$s3,$s4      #k==3 so f=i-j
Exit:
```


24