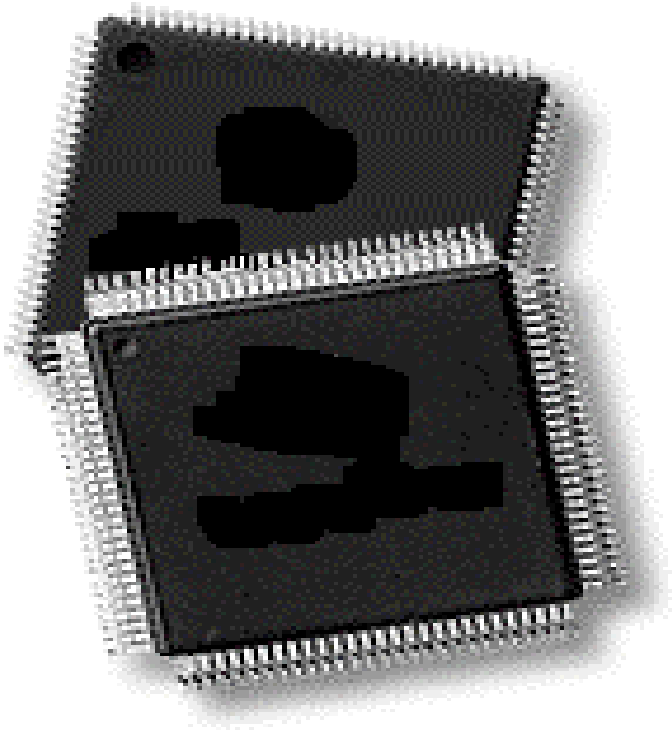
EECS 322: Computer Architecture

The SPIM simulator

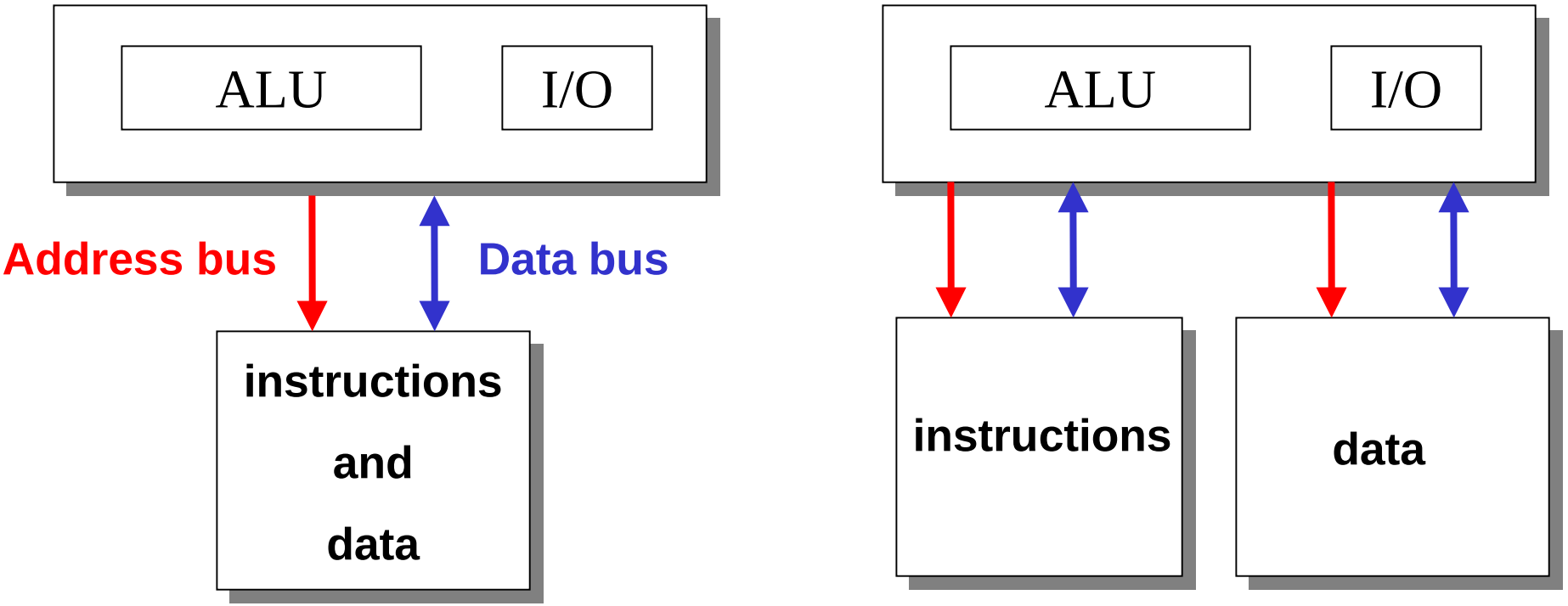


Website and Homework

Homework Website (temporary)
http://sfo.ces.cwru.edu

Problems from book (427-432)
5.1-2, 5.5-6, 5.9, 5.14-18, 5.24

Von Neuman & Harvard Architectures (PH p. 35)

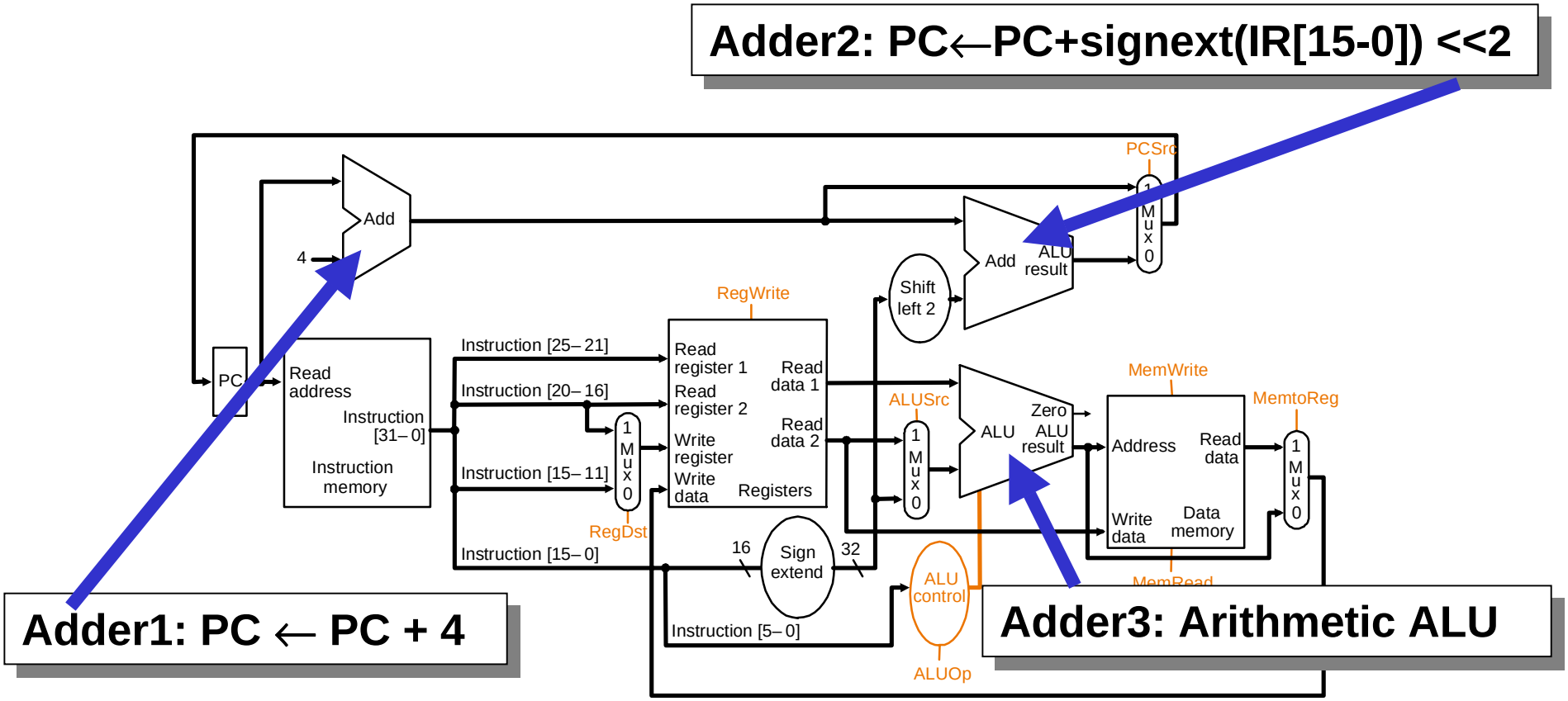


Von Neuman architecture **Area efficient but requires higher bus bandwidth because instructions and data must compete for memory.**

Harvard architecture **was coined to describe machines with separate memories. Speed efficient: Increased parallelism.**

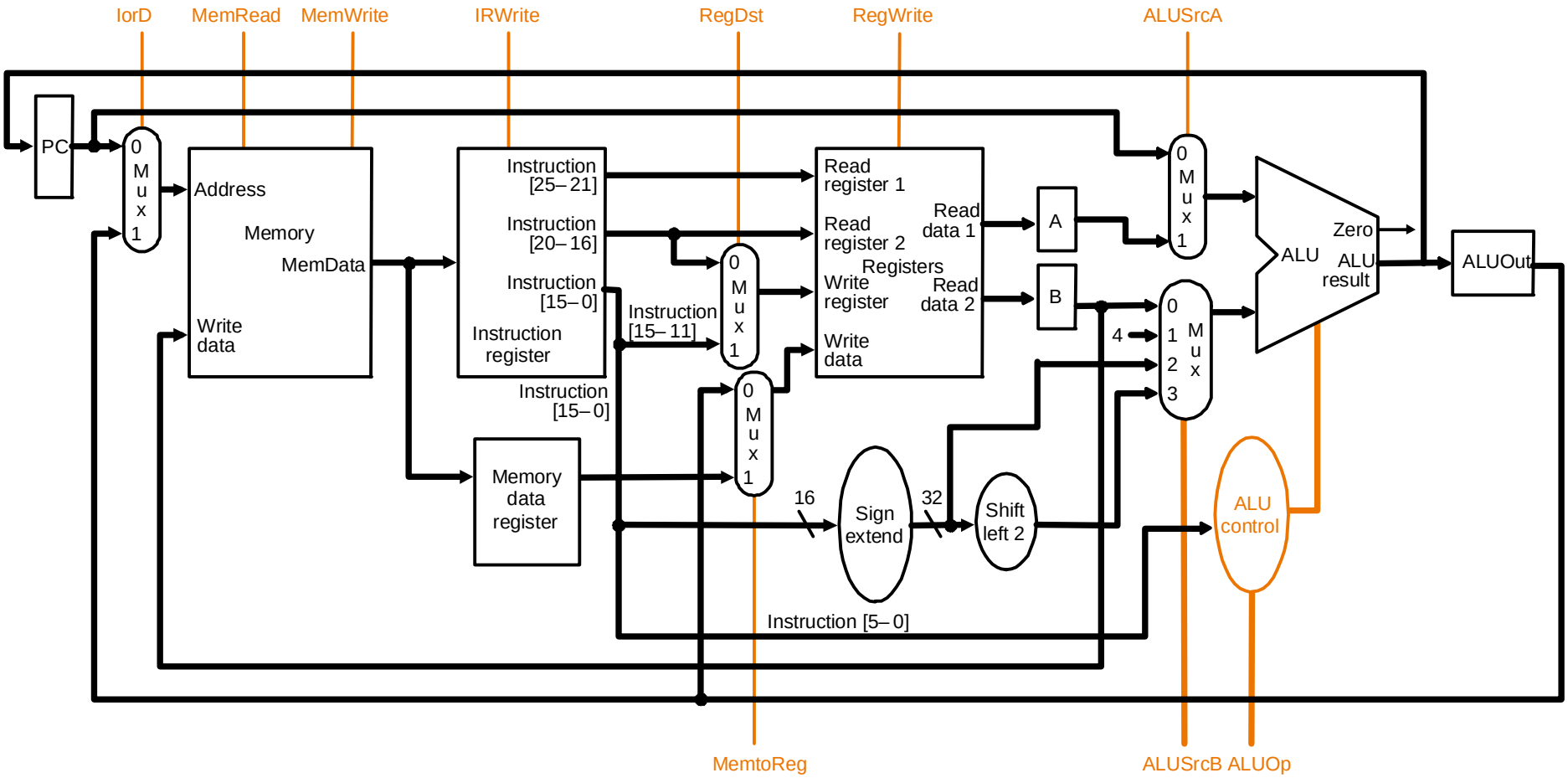
Recap: Single Cycle Implementation

- Calculate instruction cycle time assuming negligible delays except:
  - memory (2ns), ALU and adders (2ns), register file access (1ns)



Single Cycle = 2 adders + 1 ALU + 4 muxes

Multi-cycle Datapath



Multi-cycle = 5 Muxes + 1 ALU + Controller + 4 Registers (A,B,MDR,ALUOut)

Single-cycle = 4 Muxes + 1 ALU + 2 adders

Single/Multi-Clock Comparison

add = 6ns = Fetch(2ns)+RegR(1ns)+ALU(2ns)+RegW(2ns)

lw = 8ns = Fetch(2ns)+RegR(1ns)+ALU(2ns)+MemR(2ns)+RegW(2ns)

sw = 7ns = Fetch(2ns)+RegR(1ns)+ALU(2ns)+MemW(2ns)

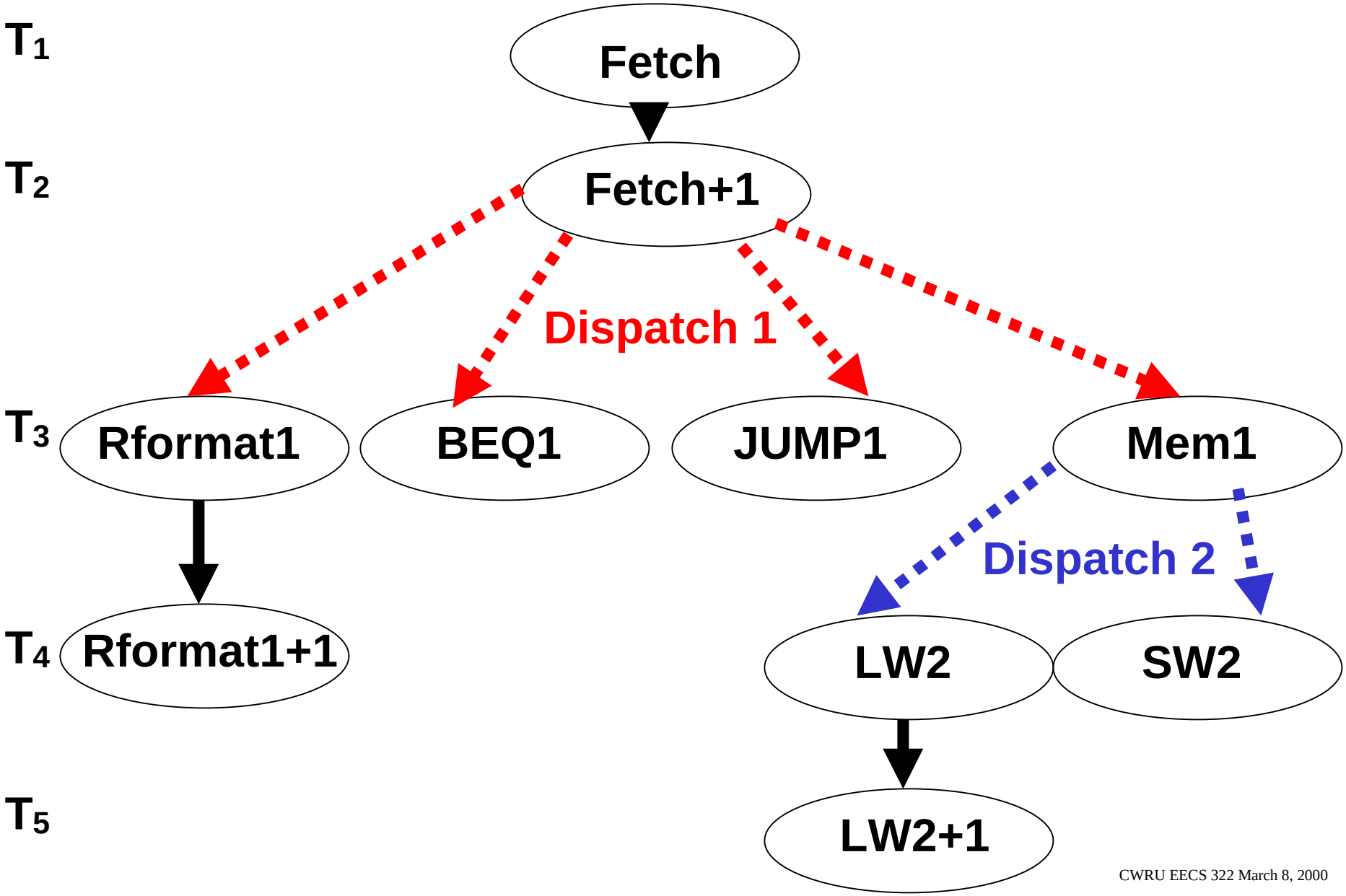
beq = 5ns = Fetch(2ns)+RegR(1ns)+ALU(2ns)

j = 2ns = Fetch(2ns)

$$\frac{\text{CPU single}-\text{cycle clock}}{\text{CPU multi}-\text{cycle clock}} = \frac{8ns}{6.3ns} = 1.27 \text{ times faster}$$

Architectural improved performance without speeding up the clock!

Microprogramming: program overview



The Spim Simulator

Spim download: ftp://ftp.cs.wisc.edu/pub/spim

unix: ftp://ftp.cs.wisc.edu/pub/spim/spim.tar.gz

win95: ftp://ftp.cs.wisc.edu/pub/spim/spimwin.exe

Spim documentation

Main document

Appendix A.9 SPIM Patterson & Hennessy pages A-38 to A75

ftp://ftp.cs.wisc.edu/pub/spim/spim_documentation.ps

ftp://ftp.cs.wisc.edu/pub/spim/spimwin.ps

Spim runnable code samples (Hello World.s, simplecalc.s, ...)

http://vip.cs.utsa.edu/classes/cs2734s98/overview.html

Other useful links

http://www.cs.wisc.edu/~larus/spim.html

http://www.cs.bilkent.edu.tr/~baray/cs224/howspim1.html

MIPS registers and conventions

Name	Number	Conventional usage
$0	0	Constant 0
$v0-$v1	2-3	Expression evaluation & function results
$a0-$a3	4-7	Arguments 1 to 4
$t1-$t9	8-15,24,35	Temporary (not preserved across call)
$s0-$s7	16-23	Saved Temporary (preserved across call)
$k0-$k1	26-27	Reserved for OS kernel
$gp	28	Pointer to global area
$sp	29	Stack pointer
$fp	30	Frame pointer
$ra	31	Return address (used by function call)

MIPS Register Name translation

 # calculate f = (g + h) - (i + j) (PH p. 109, file: simplecalc.s)

Assember .s		Translated (1 to 1 mapping)		
addi	$s1, $0, 5	addi	$17, $0, 5	#g = 5
addi	$s2, $0, -20	addi	$18, $0, -20	#h = -20
addi	$s3, $0, 13	addi	$19, $0, -20	#i = 13
addi	$s4, $0, 3	addi	$20, $0, 3	#j = 3
add	$t0, $s1, $s2	add	$8, $17, $18	#$t0=g + h
add	$t1, $s3, $s4	add	$9, $19, $20	#$t1=i + j
sub	$s0, $t0, $t1	sub	$16, $8, $9	#f=(g+h)-(i+j)

System call 1: print_int $a0

- System calls are used to interface with the operating system to provide device independent services.

- System call 1 converts the binary value in register $a0 into ascii and displays it on the console.

- This is equivalent in the C Language: printf("%d", $a0)

Assember .s		Translated (1 to 1 mapping)		
li	$v0, 1	ori	$2, $0, 1	#print_int (system call 1)
add	$a0,$0,$s0	add	$4,$0,$16	#put value to print in $a0
syscall		syscall		

System Services

Service	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		$v0=integer
read_float	6		$f0=float
read_double	7		$f0=double
read_string	8	$a0=buf, $a1=len	
sbrk	9	$a0=amount	$v0=address
exit	10		

System call 4: print_string $a0

- System call 4 copies the contents of memory located at $a0 to the console until a zero is encountered
- This is equivalent in the C Language: printf("%s", $a0)

Assember .s

```
.data
.globl msg3
msg3:    .asciiz "\nThe value of f is: "
.text
li       $v0, 4
la       $a0,msg3
syscall
```

Translated (1 to 1 mapping)

```
ori    $2, $0, 4    #print_string
lui    $4,4097      #address of string
syscall
```

Note the "z" in asciiz

msg3 is just a label but must match

.asciiz data representations

.data: items are place in the data segment

which is not the same the same as the .text segment !

Assember .s

```
msg3:       .asciiz "\nThe va"
```

Same as in assembler.s

```
msg3:       .byte  '\n','T','h', 'e', ' ', 'v', 'a',  0
```

Same as in assembler.s

```
msg3:       .byte  0x0a, 0x54, 0x68, 0x65
            .byte  0x20, 0x76, 0x61, 0x00
```

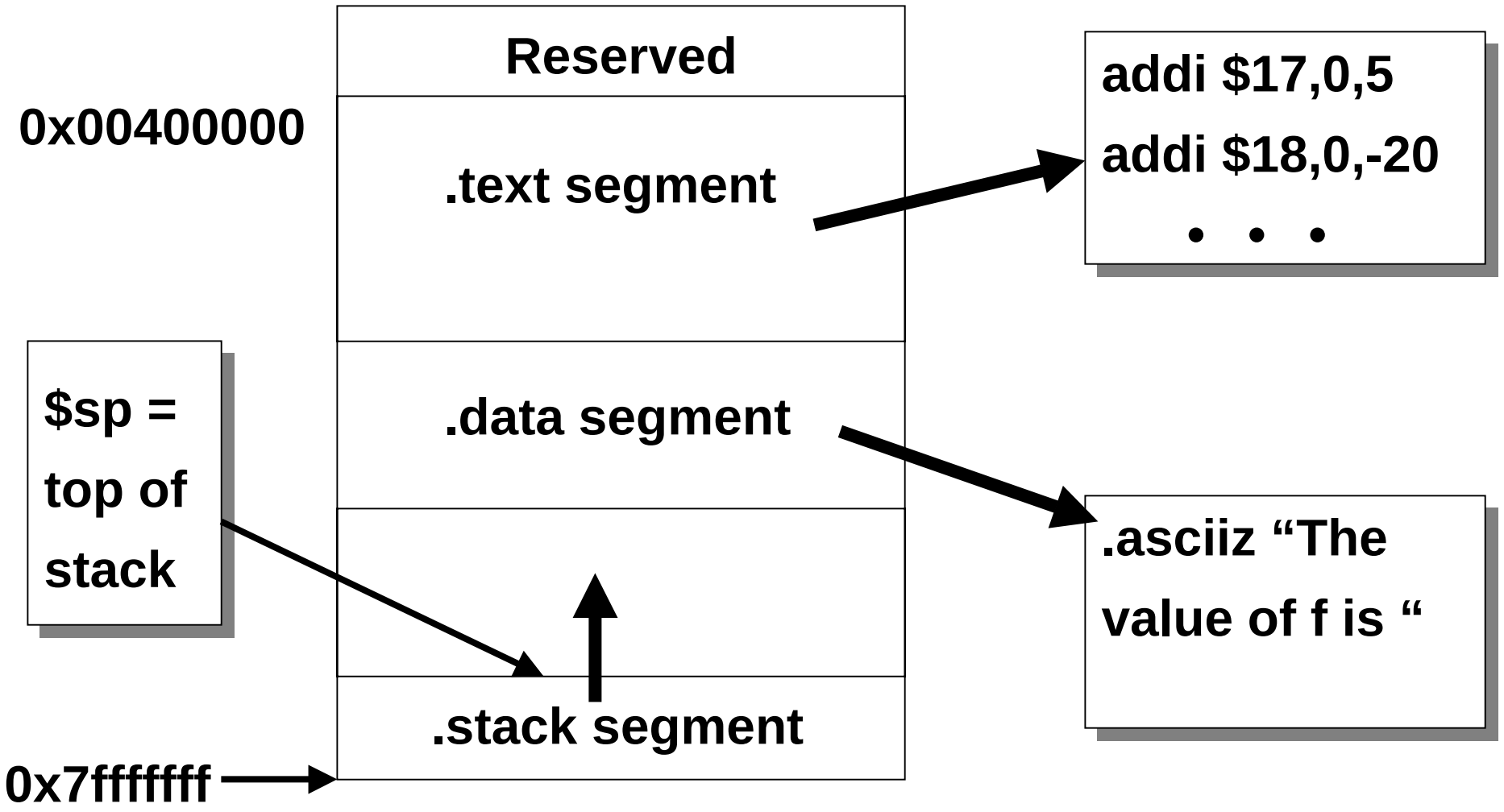
Same as in assembler.s

```
msg3:       .word 0x6568540a,  0x00617620
```

Big endian format

Translated in the .data segment: 0x6568540a 0x00617620

Memory layout: segments



- Segments allow the operating system to protect memory
- Like Unix file systems: .text Execute only, .data R/W only

Hello, World: hello.s

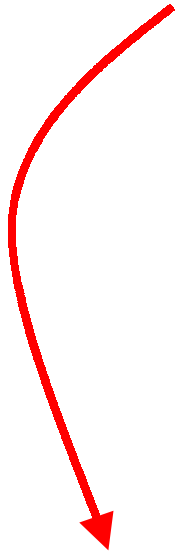
```
# main( ) {
#         printf("\nHello World"\n");
# }
        .globl main
main:                                   #main has to be a global label
        addu    $s7, $0, $ra            #save the return address in a global reg.
        .data
        .globl  hello
hello:  .asciiz "\nHello World\n"       #string to print
        .text
        li      $v0, 4                  # print_str (system call 4)
        la      $a0, hello              # $a0=address of hello string
        syscall

        # Usual stuff at the end of the main
        addu    $ra, $0, $s7            #restore the return address
        jr      $ra                     #return to the main program
        add     $0, $0, $0              #nop
```

Note: alternating .text, .data, .text

Simplecalc.s (PH p. 109)

Order of .text and .data not important

```
        .globl main
main:   addu    $s7, $0, $ra        #save the return address
        addi    $s1, $0, 5          #g = 5
        addi    $s2, $0, -20        #h = -20
        addi    $s3, $0, 13         #i = 13
        addi    $s4, $0, 3          #j = 3
        add     $t0, $s1, $s2       #register $t0 contains g + h
        add     $t1, $s3, $s4       #register $t1 contains i + j
        sub     $s0, $t0, $t1       #f = (g + h) - (i + j)
        li      $v0, 4              #print_str (system call 4)
        la      $a0, message        # address of string
        syscall
        li      $v0, 1              #print_int (system call 1)
        add     $a0, $0, $s0        #put value to print in $a0
        syscall
        addu    $ra, $0, $s7        #restore the return address
        jr      $ra                 #return to the main program
        add     $0, $0, $0          #nop
        .data
        .globl  message
message:        .asciiz "\nThe value of f is: "     #string to print
```



Simplecalc.s without symbols (PH p. 109)

```
                .text
0x00400020      addu    $23, $0, $31        # addu  $s7, $0, $ra
0x00400024      addi    $17, $0, 5          # addi  $s1, $0, 5
0x00400028      addi    $18, $0, -20        # addi  $s2, $0, -20
0x0040002c      addi    $19, $0, 13         # addi  $s3, $0, 13
0x00400030      addi    $20, $0, 3          # addi  $s4, $0, 3
0x00400034      add     $8, $17, $18        # add   $t0, $s1, $s2
0x00400038      add     $9, $19, $20        # add   $t1, $s3, $s4
0x0040003c      sub     $16, $8, $9         # sub   $s0, $t0, $t1
0x00400040      ori     $2, 0, 4            #print_str (system call 4)
0x00400044      lui     $4, 0x10010000      # address of string
0x00400048      syscall
0x0040004c      ori     $2, 1               #print_int (system call 1)
0x00400050      add     $4, $0, $16         #put value to print in $a0
0x00400054      syscall
0x00400058      addu    $31, $0, $23        #restore the return address
0x0040005c      jr      $31                 #return to the main program
0x00400060      add     $0, $0, $0          #nop
```

```
                .data
0x10010000      .word   0x6568540a, 0x6c617620, 0x6f206575
                .word   0x20662066, 0x203a7369, 0x00000000
```

Single Stepping

Values changes after the instruction!

$pc	$t0 $8	$t1 $9	$s0 $16	$s1 $17	$s2 $18	$s3 $19	$s4 $20	$s7 $23	$ra $31
00400020	?	?	?	?	?	?	?	?	400018
00400024	?	?	?	?	?	?	?	400018	400018
00400028	?	?	?	5	?	?	?	400018	400018
0040002c	?	?	?	5	fffffec	?	?	400018	400018
00400030	?	?	?	5	fffffec	0d	?	400018	400018
00400034	?	?	?	5	fffffec	0d	3	400018	400018
00400038	fffff1	?	?	5	fffffec	0d	?	400018	400018
0040003c	?	10	?	5	fffffec	0d	?	400018	400018
00400040	?	?	ffffffe1	5	fffffec	0d	?	400018	400018