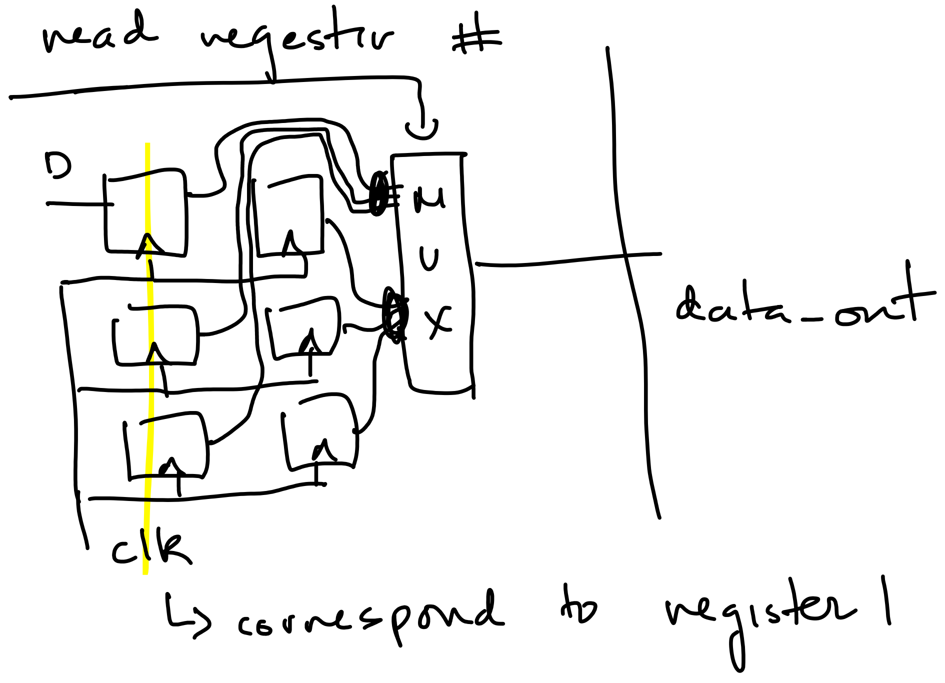
HW 1

Thursday, April 4, 2024 7:23 PM

>> 1.



>> 2.
add $3, $15, $4 # adds k + n
sub $15, $zero, $15 # inverts k
addi $15, 47 # performs 47 - k
lui $5, 0x2D
addi $15, $15, $5
lh $5, 0x7EC8($15)
sub $7, $7, $5 # performs q - p[47-k]
lui $5, 0x2D // load the upper bytes with 2D and clear out the bottom bytes
add $3, $3, $5 // add it to three
sh $7, 0x7EC8($3) # writes the result to p[n+k]

>> 3. (assuming that the bits are 0-indexed here)
slr $13, $14, 10 # right shift $14 by 10
sll $13, $13, 24 # left shift $13 by 24 (we have the desired value in the right place
andi $22, $22, 0x0FFF # zero out the first 8 bits of reigster $22
or $22, $22, $13 # or the result of $13

or $22, $22, $15 # or the result of $15

>>4.
lui $5, 414 # load 414 into the first 16 bits of $5
sw $5, 24($0) stores the word in $5 in the memory address computed by offset of 24
lbu $1, 25($0) loads unsigned byte into 1 from memory 25
lbu $2, 26($0) loads unsigned byte into 2 from memory 26

Memory	24	25	26	27
Big E	0	414	0	29
Little E	27	0	414	0

If the system is little endian, the values of 1 and 2 are 414 and 0 respectively, however, if it is bi
the value is 0 and 414 respectively.

>>5.
101000,00111,10101,0000000001001101

>>6.

address	op	rs	rt	rd	shamt	func	opcode
0000 0001 1000 0100	001000	00000	00101	0. . . 0100 1010			addi
0000 0001 1000 1000	000000	00000	00011	00111	00010	000000	sll
0000 0001 1000 1100	000000	01001	00111	00111	00000	100000	add
0000 0001 1001 0000	100011	00111	00111	0 0			lw
0000 0001 1001 0100	000100	00111	00101	0. 0101			beq
0000 0001 1001 1000	001000	00011	00011	0000 0000 0000 0001			addi
0000 0001 1001 1100	000000	00110	00111	01111	00000	101010	slt
0000 0001 1010 0000	000111	01111	00000	1 1000			bgtz
0000 0001 1010 0100	000000	00110	00111	00110	00000	100010	sub
0000 0001 1010 1000	000010	1 10110					j

(7) Add a new pseudoinstruction to the MIPS assembly language. The new pseudoinstruction is `jnerm` (jump-not-equal-register-memory): the contents of a specified register are compared to the word stored at a 32-bit address specified in the pseudoinstruction. If the values are **not equal**, a jump is performed to a 32-bit destination address specified in the pseudoinstruction. Both 32-bit addresses are assumed to be multiples of 4 (you do **not** need to check this).

For example:

```
jnerm   $5,0x345D2C,0xDABA9878
```

compares the contents of register 5 to the word stored at address 0x345D2C. If the values compared are **not equal**, a jump is performed to address 0xDABA9878.

Show an **efficient** assembly code implementation of the example above using **only** real MIPS instructions. Minimize the use of registers. You **cannot** use any labels in your assembly code.

State clearly **every** assumption you make.

- Instructions are 32 bits long
- Assume that we can not use 64-bit registers

```
lui $2, 0x34
ori $2, 0x5D2C
```
$\Big\}$ reading value @ 0x345D2C

```
lw   $2, 0($2)
beq  $2, $5, 4
```
— comparing against $5

```
lui $2, 0xDABA
ori $2, 0x9878
```
$\Big\}$ performing jump.

```
jr   $2
```

implementation only uses one register $2 and does not modify $5.

(8) Write a MIPS assembly program that will produce different results depending on whether the processor is big endian or little endian. Specifically, your program must store the value 0 into the byte at address 149 if the processor is little endian but store the value 1 into the byte at address 149 if the processor is big endian. If necessary, your program may modify bytes 6-13 in main memory as well as registers $7, $8, and $9. Your program may not modify any other memory locations or registers.

	6,	7
big	1	0
little	0	1

store halfword @ addr. 6, then read from addr 7.

```
addi  $7, $0, 0x0100
sh    $7, 6($0)
lb    $7, 6($0)
sb    $7, 149($0)
```