Assignment 1 section: 09

① CSE340 G.M. ~~Arafat~~ Rahman 1910498

1. In a code (in C/Java/Python etc) we have multiple of lines each containing some operations. When these codes get converted into MIPS Instruction (can be R or I J type), the cpu needs to keep track which line of instruction is getting executed and what will be executed next. For this task cpu has a register called program counter. It holds the address of current instruction. On the other side MIPS register 0 or $zero is the constant 0, so $0 register can only hold 0, it can not be overwritten. Beside this, we can use $0 register in various instructions. For instance, for moving values between register. add $t0, $s0, $zero. But program counter can not be used in any type of instructions.

(2)

Main memory contains portions or slots of 8 bits.

So, for sequential instruction execution:

$\frac{64}{8} = 8$, and $\frac{128}{8} = 16$ will be the increment.

2. We know, lw $4, increment x offset (base address)
↓ destination

increment for 128-bit architecture, $128/8 = 16$

offset in A[5] is = 5.

then, x = increment x offset

= 16 x 5 = 80 (Ans)

(3)

$X \rightarrow \$s0$
$Y \rightarrow \$s1$
$Z \rightarrow \$s2$

3. $X = 3Y + 5Z + 10$

$3Y \Rightarrow$ sll $t0, \$s1, 1$ — R-type

000 000	00000	$s1, $17 10001	$t0, $8 01000	00001	XXXXXX
opcode	rs	rt	rd	shamt	funct

add $t0, $t0, $s1 — R-type (3Y)

000000	$t0, $8 01000	$s1 10001	$t0, $8 01000	00000	XXXXXX
opcode	rs	rt	rd	shamt	funct

$5Z \Rightarrow$ sll $t1, \$s2, 3$ — R-type

000000	00000	$s2, $18 10010	$t1, $9 01001	00010	XXXXXX
opcode	rs	rt	rd	shamt	funct

add $t1, t1, $s2 — R-type (5Z)

000000	$t1 01001	$s2 10010	$t1 01001	00000	XXXXXX
opcode	rs	rt	rd	shamt	funct

add $t0, $t0, $t1 — R-type (3Y+5Z)

000000	$t0 01000	$t1 01001	$t0 01000	000000	XXXXXX

④

addi $s0, $t0, 10 — I-type

xxxxxx	$t0 01000	$s0 10000	0000 0000 0000 1010
opcode	rs	rt	offset

for: Y = 4X + Arr[10] + 2Z

for 2Z ⇒ sll $t0, $s2, 1 — R-type

000 000	0000 0	$s2 10010	$t0 01000	0000 1	funct xxxxxx
opcode	rs	rt	rd	shamt	

for 4X ⇒ sll $t1, $s0, 2 — R-type

000 000	0000 0	$s0 10000	$t1 01001	0000 1	xxxxxx
opcode	rs	rt	rd	shamt	funct

for Arr[10] ⇒

lw $t2, 4x10 ($s4) — I-type

xxxxxx	$s4 10100	$t3 01010	0000 0000 0010 1000
opcode 6bits	rs 5bits	rt	offset

(5)

4x + Arr[10] ⇒ add $t1, $t1, $t2 R-type

000 000	$t1 01001	$t2 01010	$t1 01001	00000	xxx xxx
opcode	rs	rt	rd	shamt	funct

Now, Y = 4x + Arr[10] + 2Z ⇒

add $s1, $t1, $t0 — R-type

000 000	$t1 01001	$t0 01000	$s1 10001	00000	xxx xxx
opcode	rs	rt	rd	shamt	funct

Now, Z = 5X + 5Y + Z

5X ⇒ sll $t0, $s0, 2 R-type

000 000	00000	$s0 10000	$t0 01000	00010	xxx xxx
opcode	rs	rt	rd	shamt	funct

add $t0, $t0, $s0 — R type

000 000	$t0 01000	$s0 10000	$t0 01000	00000	xxx xxx
opcode	rs	rt	rd	shamt	funct

(6)

sll $t1, $s1, 2 R-type

000 000	0000 0	$s1 / 10001	$t1 01001	0010	xxx xxx
opcode	rs	rt	rd	shamt	funct

add $t1, $t1, $s1 R-type

000 000	$t1 01001	$s1 10001	$t1 01001	000 00	xxx xxx
opcode	rs	rt	rd	shamt	funct

sub $t0, $t0, $t1 R-type

000 000	$t0 01000	$t1 01001	$t0 01000	000 00	xxx xxx
opcode	rs	rt	rd	shamt	funct

Now, Z = 5X + 5Y + Z

add $s2, $t0, $s2 R-type

10010 10010

000 000	$t0/01000	$s2	$s2	000 00	xxx xxx
opcode	rs	rt	rd	shamt	funct

(7)

Now, Arr[10] = X+Y-30;

X+Y ⇒ add $t0, $s0, $s1 R-type

000 000	$s0 10000	$s1 10001	$t0 01000	000 00	XXX XXX
opcode	rs	rt	rd	shamt	funct

X+Y-30 ⇒ addi $t0, $t0, -30 I-type

XXXXXX	$t0 01000	$t0 01000	1111 1111 1110 0010
opcode	rs	rt	offset/immediate

binary of 30 in 16 bit

0000 0000 0001 1110

2's complement: 1111 1111 1110 0001 + 1

1111 1111 1110 0010 (-30 using 2's complement)

Now, Arr[10] = X+Y-30 ⇒

sw $t0, 4×10($s4) → I-type

(8)

0000 0000 0010 1000

XXX XXX	$s4 10100	$t0 01000	40 / 0000 0000 0010 1000
opcode	rs	rt	offset

4. Base of B[] → $s0 , i → $s1 , &f → $s2

f = B[i]

add $t1 , $s0, $s1

lw $s2, 0($t1)

~~add $s2, $zero, $t2~~

5. MIPS does not have an assignment ~~op~~ instruction. int a = 100. suppose a is in $s0 register. we have a $zero register. We know addi instruction can add a value in a register and from a constant integer. Here constant is 100. So 0+100 = 100. So

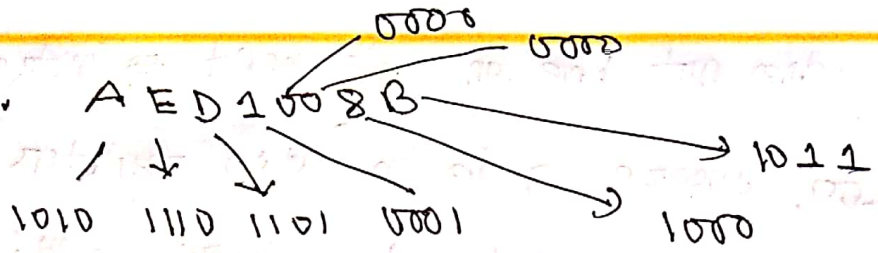
addi $s0, $zero, 100

addi → keyword
$s0 → destination
$zero → source 1
100 → source 2 (constant)

Here addi will add value in $zero with 100 and will store in register $s0 which is the register of variable a.

(10)

6. AED100 8B



We know, every 1 bit of hexadecimal is equivalent to 4 bit binary. As we know main memory is ~~so if~~ divided into 8 bit slot, then 32 bit ~~data~~ will be stored in 4 different slots in main memory. AE = 8 bit, D1 = 8 bit, 00 = 8 bit and 8B = 8 bit.

(i) Big Endian: In Big Endian, most-significant byte ~~get~~ stored in least address of a word. Here 0x0012830A is the last address of the given memory address, so in:

0x00128830A = AE ~~or~~ MSB byte will be stored

0x00128830B → D1, 0x00128830C → 00 and LSB 8B will get stored in 0x00128830D.

(11)

(ii) Little Endian:

In little Endian order, LSB byte get stored at last address of a word. So

88 BB → 0x00128304

00 B0 → 0x00128305

1D B1 → 0x00128306

EA AE → 0x00128307