Assignment #1

Bo Yao, by677
CS-GY 6133 Tianyu Gu, tg1553

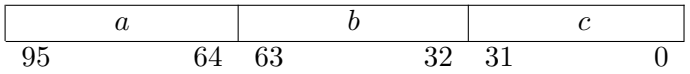
Solution 1

(a)

Since the ISA has only one instruction, there's no need to assign any bits to the opcode. The memory is 4GB large, so each operand need 32 bits to be encoded.

Therefore, the minimum number of total bits is 0 + 32*3 = 96.

subleq:



(b)

It is 3-Address ISA.

It is a memory-memory ISA.

(c)

initial:
→ $M[A] = 7$, $M[B] = -9$, $M[C] = 4$, $M[D] = 3$, Jump to L_0.

L_0: subleq C, C, L_1
→ $M[A] = 7$, $M[B] = -9$, $M[C] = \mathbf{0}$, $M[D] = 3$, Jump to L_1.

L_1: subleq D, D, L_2
→ $M[A] = 7$, $M[B] = -9$, $M[C] = 0$, $M[D] = \mathbf{0}$, Jump to L_2.

L_2: subleq A, B, L_6
→ $M[A] = \mathbf{16}$, $M[B] = -9$, $M[C] = 0$, $M[D] = 0$, Jump to *next*.

L_3: subleq D, B, L_4
→ $M[A] = 16$, $M[B] = -9$, $M[C] = 0$, $M[D] = \mathbf{9}$, Jump to *next*.

L_4: subleq C, D, L_5
→ $M[A] = 16$, $M[B] = -9$, $M[C] = \mathbf{-9}$, $M[D] = 9$, Jump to L_5.

L_5: subleq D, D, L_9
→ $M[A] = 16$, $M[B] = -9$, $M[C] = -9$, $M[D] = \mathbf{0}$, Jump to L_9.

L_9:
→ $M[A] = 16$, $M[B] = -9$, $M[C] = -9$, $M[D] = 0$, *End*.

In general, this snippet of code does the following operation:
If $M[A] > M[B]$, $M[C] \leftarrow M[B]$. If $M[A] \leq M[B]$, $M[C] \leftarrow M[A] - M[B]$. D is a temporary memory location.

(d)

L_0: subleq $M_2,\ M_2,\ L_1$	// $M_2 \leftarrow 0$
L_1: subleq $M_2,\ M_0,\ L_2$	// $M_2 \leftarrow M_2 - M_0$
L_2: subleq $M_0,\ C_1,\ L_4$	// $M_0 \leftarrow M_0 - 1$
L_3: subleq $M_1,\ M_1,\ L_1$	// if $M_0 > 0$, loop; else, end loop
L_4: subleq $M_1,\ M_2,\ L_5$	// $M_1 \leftarrow -M_2$
L_5: Exit.	// Exit. The total no. of steps is N*3+1.

Solution 2

Note: The instruction format in HW pdf is different from which in one of the reference pdf. In this solution, the format we used was referred to the "BASIC INSTRUCTION FORMATS" table in the references as following:

R:	op	rs	rt	rd	sha	func
I:	op	rs	rt	imm		
J:	op	rs	addr			

For example, instruction ADD and XORI:

$rd = rs + rt \quad \rightarrow \quad$ **add** rs, rt, rd
$rt = rs\ xor\ 1 \quad \rightarrow \quad$ **xori** $rs, rt, 1$

(a)

sub $rx,\ rx,\ rx$	// $rx \leftarrow 0$
sub $rx,\ rt,\ rx$	// $rx \leftarrow -rt$
sub $rs,\ rx,\ rd$	// $rd \leftarrow rs - (-rt)$

(b)

xori $rt,\ rt,\ 1$	// $\sim rt$
xori $rs,\ rs,\ 1$	// $\sim rs$
nor $rt,\ rs,\ rd$	// $rd \leftarrow \sim(\sim rt \mid \sim rs) = rt\ \&\ rs$
xori $rt,\ rt,\ 1$	// rt
xori $rs,\ rs,\ 1$	// rs

(c)

// MIPS uses Big-endian

lhu $rs,\ rt,\ imm$	// load higher 2 bytes to rt
lhu $rs,\ rx,\ imm + 2$	// load lower 2 bytes to rx
sll $rt,\ rt,\ 16$	// rt shift left by 16 bits (2 bytes)
or $rt,\ rx,\ rt$	// $rt \leftarrow rt \mid rx$

(d)

// In Jump instruction, *addr* is a 26 bits immediate. Let's assume that immediate *haddr* = 16b'(*addr* >> 16), *laddr* = 16b'(*addr* & $0xFF$).

ori	\$0, *rx*, *haddr*	// $rx \leftarrow haddr$
sll	*rx*, *rx*, 16	// $rx \leftarrow \{haddr,\ 16b'0\}$
ori	*rx*, *rx*, *laddr*	// $rx \leftarrow \{haddr,\ laddr\} = addr$
sll	*rx*, *rx*, 2	// $rx \leftarrow \{addr,\ 2b'0\} = addr << 2$
jr	*rx*	

// The instructions above only make $PC = (addr << 2)$. However, **j** *addr* $\rightarrow PC = PC\&0xF0000000|(addr << 2)$. But there's no instructions to get the value of PC. So the higher 4 bits of PC are lost.
// Instruction j is simply equivalent to jal, if it's ok to ignore \$ra.

Solution 3

(a)

3-address: **ADD** *Rd*, *Rs*, *Rn*
2-address: **AND** *Rd*, *Rs*

(b)

Supported:

- Displaced/Based

  LDR *Rd*, [*Rb*, $\#Imm$] // $\#Imm$ is a 5 bits offset, $Rd \leftarrow Mem[Rb + Imm]$
  STR *Rd*, [*Rb*, $\#Imm$] // $\#Imm$ is a 5 bits offset, $Mem[Rb + Imm] \leftarrow Rd$

- Indexed

  LDR *Rd*, [*Rb*, *Ro*] // $Rd \leftarrow Mem[Rb + Ro]$
  STR *Rd*, [*Rb*, *Ro*] // $Mem[Rb + Ro] \leftarrow Rd$

Unsupported:

- Absolute

  But We can emulate an absolute addressing instruction by using the displaced/based addressing instruction with *Rb* pre-assigned to 0. But the Imm is only 5 bits long, which means we can only absolutely access to the memory locating from 0 to 2^5.

  LDR *Rd*, [\$0, $\#Imm$] // $Rd \leftarrow Mem[Imm]$
  STR *Rd*, [\$0, $\#Imm$] // $Mem[Imm] \leftarrow Rd$

- Register Indirect

  The same as above, the register indirect instruction can be emulated.

  LDR *Rd*, [*Rb*, #0] // $Rd \leftarrow Mem[Rb]$
  STR *Rd*, [*Rb*, #0] // $Mem[Rb] \leftarrow Rd$

- Auto-increment

  We are not sure whether the sum of Rb and Imm in displaced/based addressing instruction is written back to Rb or not. If the sum is written back to Rb, the auto-increment addressing mode should be considered as supported. But it's a pre-indexed addressing mode, which is different from the post-index addressing mode discussed in the class.

  $LDR \quad Rd,\ [Rb,\ \#1] \qquad // \ Rd \leftarrow Mem[Rb]$
  $STR \quad Rd,\ [Rb,\ \#1] \qquad // \ Mem[Rb] \leftarrow Rd$

- Memory Indirect