Hardware Design

Tutorial 3

Instructor: Dr. Haiyu Mao
TA: Zihao Pu

06.02.2026

Concept Questions

- (a) Why must the single-cycle CPU's clock match the slowest instruction?

  (a) In a single-cycle CPU, **every instruction completes in one cycle**.
  Thus, the clock must be long enough to accommodate the slowest instruction (e.g., load/store).

- (b) Give two advantages of the multi-cycle model compared to single-cycle.

  (b) Multi-cycle advantages:
  Shorter clock cycle ⟶ faster clock frequency.
  Hardware reuse ⟶ fewer ALUs, less hardware complexity.

KING'S College LONDON

(a) Why must the single-cycle CPU's clock match the slowest instruction?
(b) Give two advantages of the multi-cycle model compared to single-cycle.

✅ **Solution**

(a) In a single-cycle CPU, **every instruction completes in one cycle**.
Thus, the clock must be long enough to accommodate the slowest instruction (e.g., load/store).
(b) Multi-cycle advantages:
Shorter clock cycle → faster clock frequency.
Hardware reuse → fewer ALUs, less hardware complexity.

- Translate the following into MIPS assembly:

```
if (a != b)
    c = a + b;
else
    c = a - b;
```

Giving a = $s0, b = $s1, c = $s2.

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three register operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three register operands
	add immediate	addi $s1,$s2,20	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	sw $s1,20($s2)	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	lh $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	sh $s1,20($s2)	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	lb $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	lbu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	sb $s1,20($s2)	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	sc $s1,20($s2)	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	lui $s1,20	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,20	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1!= $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	set on less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	slti $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	sltiu $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

KING'S College LONDON

3

Translate the following into MIPS assembly:

```
if (a != b)
    c = a + b;
else
    c = a - b;
```

a = $s0, b = $s1, c = $s2.

Answer:

```
MAIN:
    bne  $s0, $s1, DIFF
    sub  $s2, $s0, $s1     # else: c = a - b
    j    END


DIFF:
    add  $s2, $s0, $s1     # if: c = a + b
END:
```

Other way of writing it:

```
MAIN:
  BEQ $s0, $s1, ELSE //if a == b
  ADD $s2, $s1, $s0 // execute if a !=b
  J END

ELSE:
  SUB $s0, $s1, $s2 // execute if a == b

END:
```

Code compilation to LC3 Instruction

```
// Returns the sum of numbers from 1 to n.
int sum_to_n(int n) {
    int acc = 0;
    for (int i = 1; i <= n; i++) {
        acc = acc + i;
    }
    return acc;
}


int main() {
    int n = 4;
    int s = sum_to_n(n);
    return 0;
}
```

```
MAIN
        LD   R6, STACK_TOP
        AND  R1, R1, #0
        ADD  R1, R1, #4
        JSR  sum_to_n
        ST   R0, RESULT
        HALT
```

```
sum_to_n
        ADD  R6, R6, #-1
        STR  R1, R6, #0
        AND  R0, R0, #0
        ADD  R2, R0, #1
LOOP
        NOT  R3, R1
        ADD  R3, R3, #1
        ADD  R3, R2, R3
        BRp  DONE
        ADD  R0, R0, R2
        ADD  R2, R2, #1
        BRnzp LOOP

DONE
        LDR  R1, R6, #0
        ADD  R6, R6, #1
        RET

RESULT      .BLKW 1
STACK_TOP   .FILL x4000
```

KING'S College LONDON


```
// Returns the sum of numbers from 1 to n.
int sum_to_n(int n) {
    int acc = 0;
    for (int i = 1; i <= n; i++) {
        acc = acc + i;
    }
    return acc;
}


int main() {
    int n = 4;
    int s = sum_to_n(n);
    return 0;
}

// answer

        .ORIG x3000
```

```
;==============================
; main()
;==============================
MAIN
      LD   R6, STACK_TOP       ; initialize SP


      AND  R1, R1, #0
      ADD  R1, R1, #4          ; argument n = 4 in R1


      JSR  sum_to_n            ; call


      ST   R0, RESULT          ; save returned value
      HALT



;============================================================
; sum_to_n(n)
; Argument: R1 = n
; Return : R0 = sum
;
; Only push R1 because:
;   - R1 holds n (caller needs it)
;   - We will use R1 as a temp loop var later
;============================================================
sum_to_n
      ; Save R1 on stack
      ADD  R6, R6, #-1
      STR  R1, R6, #0


      ; Locals in registers:
      ;   R0 = acc
      ;   R2 = i
      ;   R3 = (temporary for compare)


      AND  R0, R0, #0          ; acc = 0
      ADD  R2, R0, #1          ; i = 1


LOOP
```

```
        ; if (i > n) break
        NOT  R3, R1
        ADD  R3, R3, #1         ; R3 = -n
        ADD  R3, R2, R3         ; R3 = i - n
        BRp  DONE


        ; acc += i
        ADD  R0, R0, R2


        ; i++
        ADD  R2, R2, #1
        BRnzp LOOP


DONE
        ; Restore original R1 (argument register)
        LDR  R1, R6, #0
        ADD  R6, R6, #1


        RET



;==============================
; Data
;==============================
RESULT      .BLKW 1
STACK_TOP   .FILL xFDFF


        .END
```