

# 計算機組織

Lab 0

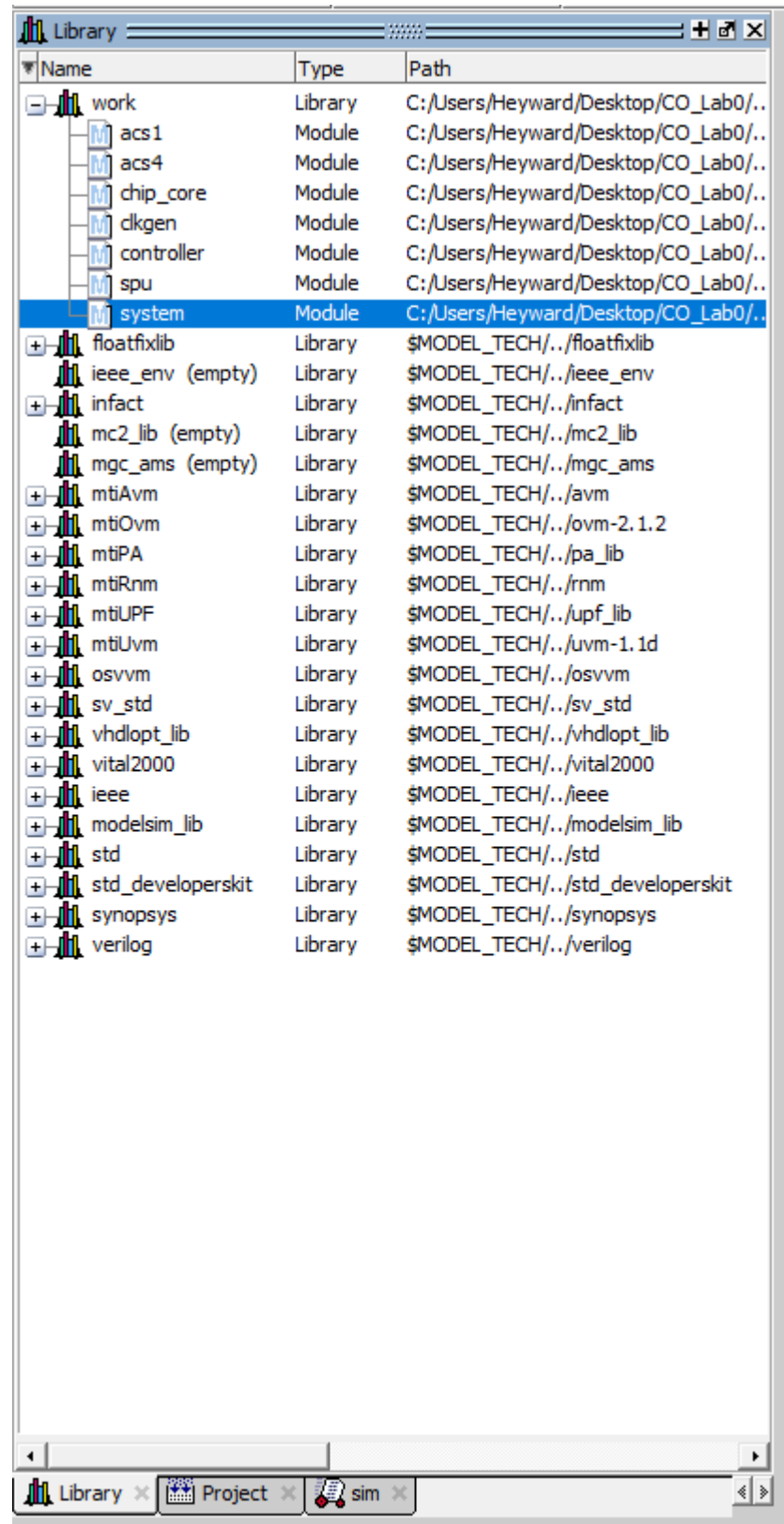
系級：土木 4B

學號：0511330

姓名：劉紘華

# Part1

## 1.Add Files into work directory.



## 2.Compile All Project Files.



The screenshot shows a project window titled "Project - C:/Users/Heyward/Desktop/CO\_Lab0/Lab0". It contains a table of project files with columns for Name, Status, Type, Order, and Modified. The file "chip\_core.v" is selected. Below the table is a transcript window showing the command "VSIM 4> run -over" and the output "# Next activity is in 15 ns."

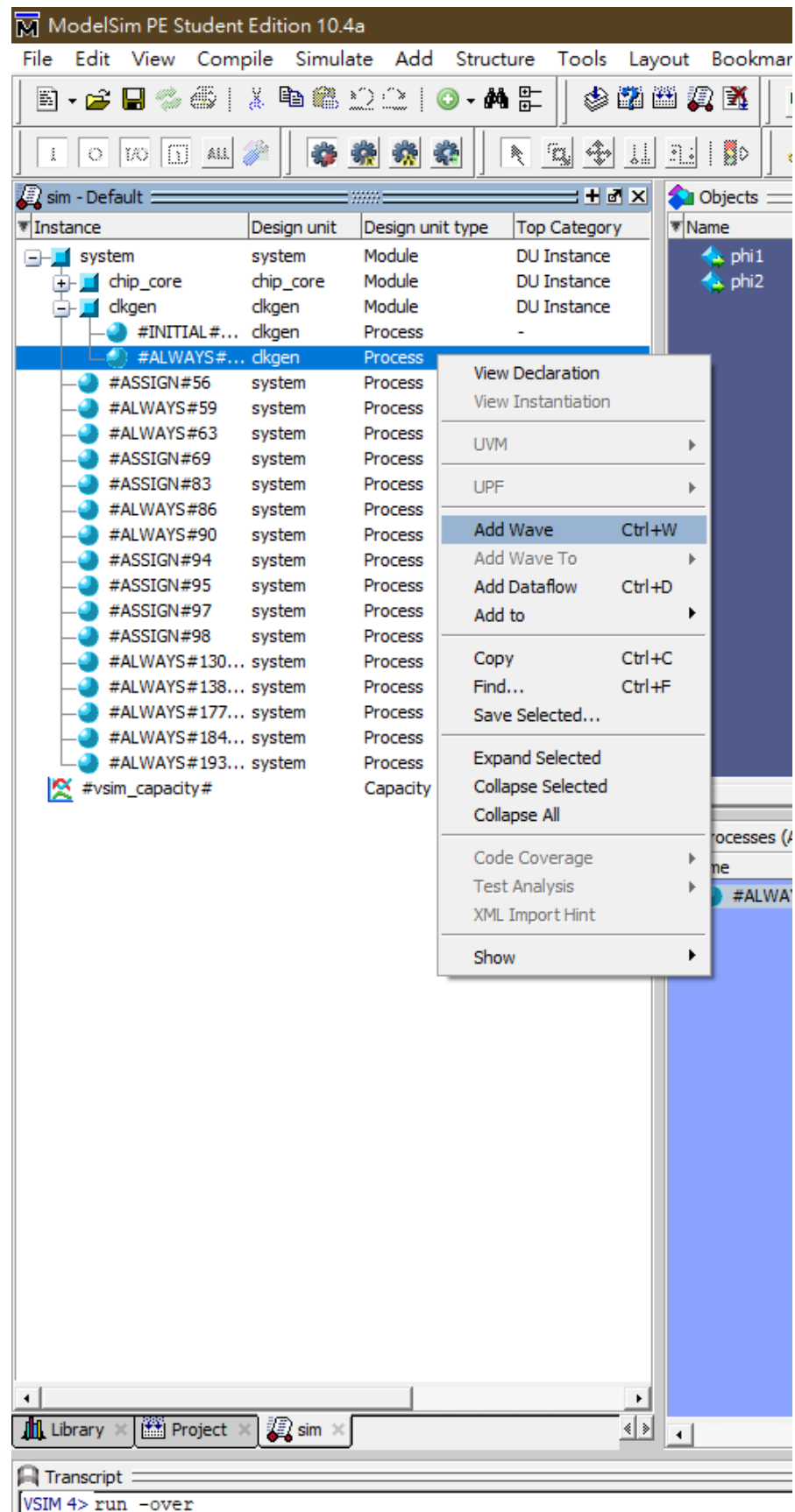
Name	Status	Type	Order	Modified
spu.v	✓	Verilog	5	03/03/2020 07:35:34 ...
controller.v	✓	Verilog	4	03/03/2020 07:35:26 ...
clkgen.v	✓	Verilog	3	03/03/2020 07:35:26 ...
chip_core.v	✓	Verilog	2	03/03/2020 07:35:26 ...
acs1.v	✓	Verilog	0	03/03/2020 07:35:26 ...
system.v	✓	Verilog	6	03/03/2020 07:35:34 ...
acs4.v	✓	Verilog	1	03/03/2020 07:35:26 ...

Library x Project x sim x

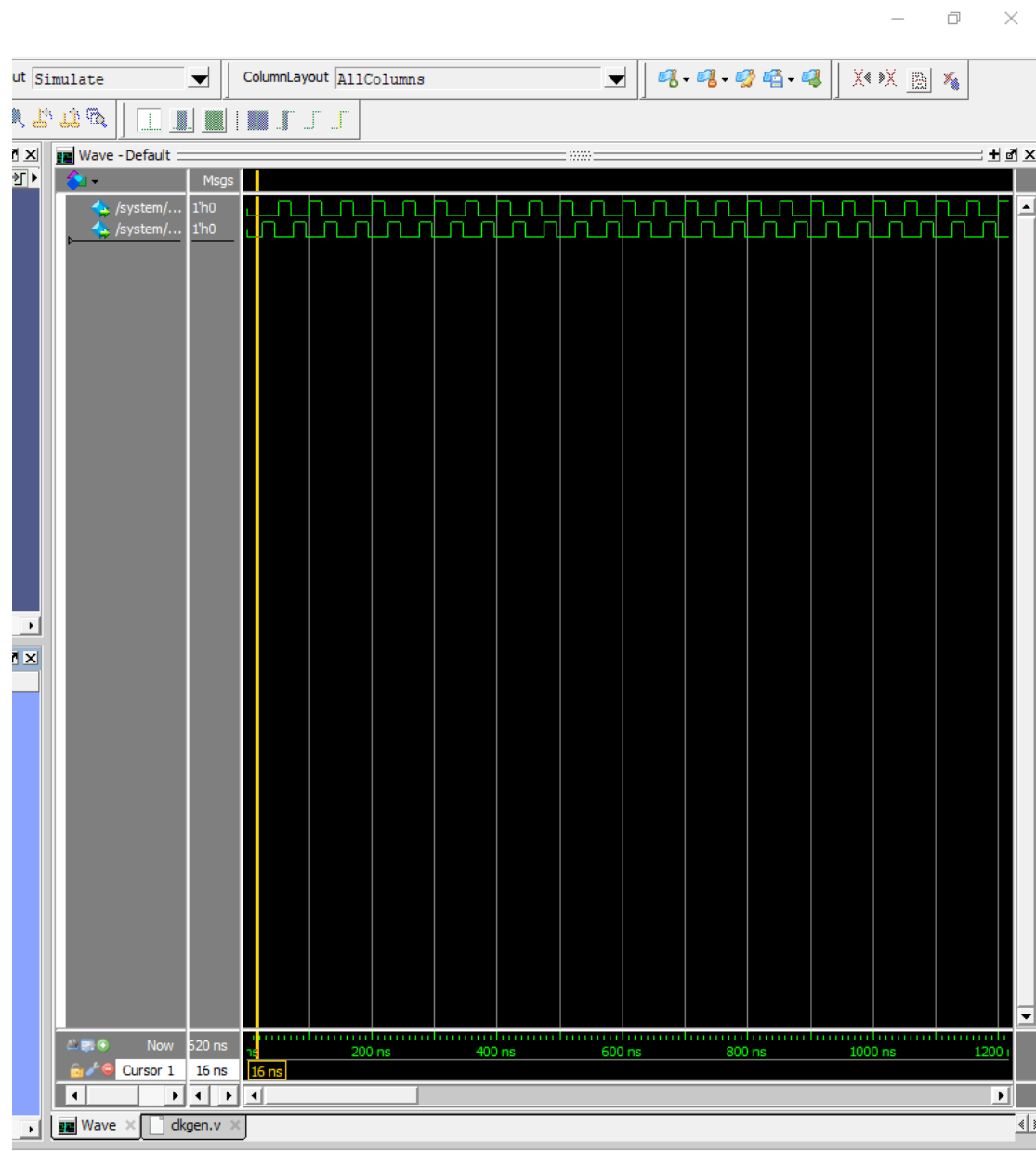
Transcript

```
VSIM 4> run -over  
# Next activity is in 15 ns.
```

### 3.Add Wave in Simulation windows.



4. Press F9 to Run the simulation and get a Wave diagram.



# Part2

## C Code:

```
int multiplication(int a, int b){
    return a*b;
}
int main(){
    int x1, x2, x3;
    x1 = 2;
    x2 = 3;
    x3 = multiplication(x1,x2);
}
```

## RISC-V 64 Code:

---

```
(1) multiplication(int, int):  # @multiplication(int, int)
(2) addi sp, sp, -32 // Add immediate value -32, sp = sp + (-32)
(3) sd ra, 24(sp) // Store doubleword, Memory [sp + 24] = ra
(4) sd s0, 16(sp) // Store doubleword, Memory [sp + 16] = s0
```

(2)把 stack pointer 往下移 4 個 doubleword 用來儲存，  
(3)和(4)分別把 return address (ra)和 stack pointer (s0)先存起來，避免執行 multiplication( )時，覆蓋掉 main( )原本存的 ra 和 s0。

---

```
(5) addi s0, sp, 32 // Add immediate value 32, s0 = sp + 32
(6) add a2, zero, a1 // Add zero, a2 = 0 + a1
(7) add a3, zero, a0 // Add zero, a3 = 0 + a0
(8) sw a0, -20(s0) // Store word, Memory[s0 - 20] = a0
(9) sw a1, -24(s0) // Store word, Memory[s0 - 24] = a1
```

(5) 利用 frame pointer (s0)指向一開始的 sp 當作 end\_ptr 使用，類似陣列中存在兩個指標 start\_ptr 和 end\_ptr。

(6)和(7)先把 function arguments 先傳遞給其他 Reg.，a1 給 a2、a0 給 a3。

(8)和(9)再把 a0 & a1 存進 Stack 中，因為 a0 和 a1 用來存放 function arguments 和 return value，先存放是為了避免被後面產生的 return value 覆蓋掉。

註：a0 = x10, a1 = x11

---

```
(10) lw    a0, -20(s0) // Load word, a0 = Memory[s0 - 20]
(11) lw    a1, -24(s0) // Load word, a1 = Memory[s0 - 24]

(12) mulw  a0, a0, a1 // a0[0:63] = Sign_Extend ( a0[0:31] * a1[0:31] )
    // RISCv 64 才有的指令，將 a0 的 0~32 位 乘以 a1 的 0~32 位，
    // 再把算出來的結果 Sign_Extend 成 64-bit，存進 a0。
```

(10)和(11)從 Stack 中把傳進來 a0, a1 存取回來 (a0 = a = 2, a1 = b = 3)  
(12)則是把 a0\*a1 的結果存進 a0 當成 **return value** (return a\*b in C code)

---

```
(13) ld    s0, 16(sp) // Load doubleword, a0 = Memory[sp + 16]
(14) ld    ra, 24(sp) // Load doubleword, ra = Memory[sp + 24]
(15) addi  sp, sp, 32 // Add immediate value, sp = sp + 32
(16) ret
```

(13)和(14)將存在 Stack 中的 s0 和 ra 存取回來  
(15)把 stack pointer 指回原來的 sp，release(2)占用的 4 個 doubleword  
(16) return 回 main

---

```
(17) main:                # @main
(18) addi  sp, sp, -32 // Add immediate value, sp = sp - 32
(19) sd    ra, 24(sp) // Store doubleword, Memory[sp + 24] = ra
(20) sd    s0, 16(sp) // Store doubleword, Memory[sp + 16] = s0
```

(18)~(20)意同 multiplication 函式裡面的(2)~(5)，把 ra 和 s0 存進 Stack 中。

---

```
(21) addi  s0, sp, 32 // Add immediate value, s0 = sp + 32
(22) addi  a0, zero, 2 // Add immediate value, a0 = 0 + 2
(23) sw    a0, -20(s0) // Store word, Memory[s0 - 20] = a0
(24) addi  a0, zero, 3 // Add immediate value, a0 = 0 + 3
(25) sw    a0, -24(s0) // Store word, Memory[s0 - 24] = a0
```

(21) 意同(5)，建立 `end_ptr`。

(22)~(24) 先把值存進 `Reg.` 中，再把 `Reg.` 中的值存進 `Stack` 中。

---

(26) `lw a0, -20(s0)` // Load word, `a0 = Memory[s0 - 20]`

(27) `lw a1, -24(s0)` // Load word, `a1 = Memory[s0 - 24]`

(28) `call multiplication(int, int)`

(26)、(27) 把 `Stack` 中的值存進 `a0, a1`，透過(28)把參數 `a0, a1` 傳遞到 `function` 中。

---

(29) `sw a0, -28(s0)` // Store word, `Memory[s0 - 28] = a0`

(30) `mv a0, zero` // Move from zero to `a0`, `a0 = 0`

// `mv` is similar to `addi a0, zero, 0 ( a0 = zero + 0 )`

(29) 把 `function` 的 `return value` 存進 `Stack` 中，(30)再把 `a0` 初始化成 0。

---

(31) `ld s0, 16(sp)` // Load doubleword, `s0 = Memory[sp + 16]`

(32) `ld ra, 24(sp)` // Load doubleword, `ra = Memory[sp + 24]`

(33) `addi sp, sp, 32` // Add immediate value, `sp = sp + 32`

(34) `ret`

(31)、(32) 把 `main` 一開始存的 `s0 & ra` 從 `Stack` 中存回來，(33)將 `sp` 指回 `top of Stack`，釋放 `main` 一開始占用的 4 個 `doubleword`。

(34) `int main()` 的結尾 ( `return 0` in C code )。