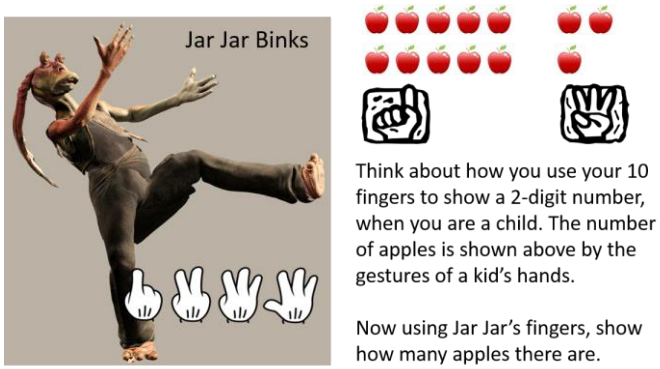
Student ID: _____ Name: _____

(Open Book – textbook and lecture slides; NO electronic device; Answers can be written in English or Chinese; Code in Assembly only unless explicitly specified)

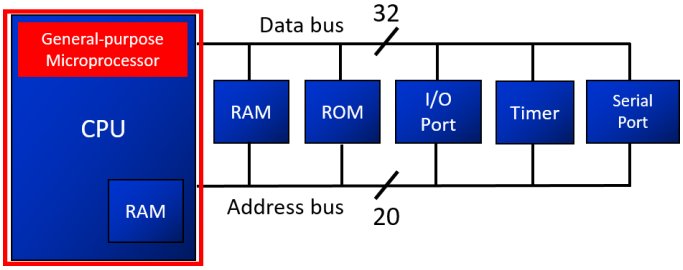
1. **[2%] [Introduction]**



ANSWER: [呂源] 15_8。

因人類小孩在學習階段依據生理的天性被教導使用兩隻手（十隻手指頭）來數數，所以當物品被記數到數目 10，因手指頭不夠使用即產生進位，所以十進位記數是自然形成的計數系統，而 Jar Jar 總共只有八隻手指頭，故數到八就會進位，所以依據前述可合理推斷他們是以 8 進位記數。
（提及八進位給 1 分，答案對給 2 分）

2. **[Introduction]** Answer the following questions briefly.



(a) **[2%]** There are two types of RAMs in the system. What kind of RAM should be in a processor?
ANSWER: [凱仁] SRAM。

(b) **[2%]** What's the differences between the RAMs in a processor and the one on the board?
ANSWER: （速度、成本各 1 分；但若寫出底線範圍的內容亦能給 1 分）
[凱仁] 主機板上的 RAM 是 DRAM（每個 cell 由 1 個電晶體構成），而處理器內的 RAM 是 SRAM（每個 cell 由 6 個電晶體構成）。DRAM 必須週期性充電，而 SRAM 則不必自動充電。速度：SRAM 快 DRAM 慢；價格：SRAM>DRAM。

(c) **[2%]** What's the maximum memory address in this system?
ANSWER: [呂源] `0xFFFFF`

(d) **[2%]** What's the data length of a general purpose register?
ANSWER: [華璽] 32 bits

(e) **[2%]** What is the bus, except address and data bus, in the system?

ANSWER: [呂源] Control Bus

(f) **[2%]** Please describe the differences between the Flash memory used for a USB flash disk and the one in an embedded system for firmware.

ANSWER: [凱仁] 嵌入式系統的 Flash Memory 是 NOR 記憶體，而 USB Flash Disk 為 NAND 記憶體。（提及 NOR、NAND 1 分，性質 1 分）

Microcontrollers use **NOR** flash, which is slower to write but permits random access.

NAND flash is used in bulk storage device and can be accessed only serially in rows.

	讀取速度	寫入速度	擦除速度	容量	成本	市佔率
NOR Flash	快	慢	慢	小	高	減少
NAND Flash	中	快	快	大	低	上升

3. **[3%] [Function / Interrupt]**

Where will be the first executed instruction located after the MSP430 is powered up?

A. `0x0000` B. `0xFFFE` C. `main(void)` D. `RSEG CODE` E. it depends

ANSWER: [凱仁] E，因為必須根據 reset vector 內所存放之位址，才知道指令從何處開始執行。

4. **[2%] [Instructions]**

How many bytes are generated for a Directive "`ORG`" after being converted to machine code by an assembler?

ANSWER: [華璽] 0 byte，`ORG` 為虛指令。

5. **[6%] [Coding]** A table with 16 elements are defined as the following. Please write the Assembly code to calculate the average by using a loop with an instruction in indirect auto-increment register addressing mode. (Only necessary code is required.)

`table: DW 0x0689,0x183C,0x5566,0x0581,……`

ANSWER: [華璽] （程式碼錯一個地方扣 1 分，扣到本題 0 分為止）

```
        mov     #table , R5
        mov     #16    , R6
        mov     #0     , R7

loop:   add.w   @R5+, R7
        dec     R6
        jnz     loop

end:
        rra     R7
        rra     R7
        rra     R7
        rra     R7
        jmp     $               ; 結果置於 R7
```

6. **[Instructions]**

(a) **[3%]** What are the differences between the instructions "`br`" and "`jmp`"?

ANSWER: [韋勳]（寫出指令長度及跳躍範圍的差別才給分）

`jmp` fits in a single word, but the range is limited to ±1KB from current location.

`br` is double word, that can go anywhere in the address space and use any addressing mode, but slower and requires an extra word of program storage.

(b) **[3%]** Which instruction is applied first when writing your code and why?

ANSWER: [華璽]（答案 1 分，原因 2 分）

先用 jmp 以節省程式使用空間，若不幸目的地超過範圍，則 assemble 結果會出現錯誤訊息，然後再修改指令為 br 即可，但是通常跳躍範圍不會太大，所以發生機會不多。

7. **[Instruction]**

How can you do a multiplication, but not using hardware multiplier of MSP430?

(a) **[3%]** Please describe the general case software approach by using "`n*11`" as an example.

ANSWER: [凱仁]（用一般解給 3 分，用暴力法、特殊解 (如：8+1+1+1) 給 1 分）

```
;; set r5 = n
mov r5, r6        ; r6 = r5
rla r5            ; r5 = n*2
add r5, r6        ; r6 = n + n*2 = n*3
rla r5            ; r5 = n*4
rla r5            ; r5 = n*8
add r5, r6        ; r6 = n*3 + n*8 = n*11
```

(b) **[5%]** Please compare and explain whether the hardware approach is better than the software one or not.

	Example	Cycles
Register → Register	`add Rs, Rd`	1 cycle

Indirect → Register	`add @Rs , Rd`
	`mov #5 , Rd`
	`mov #4 , Rd`
		2 cycles
		2 cycles
		1 cycle

Index → Register	`add S(Rs) , Rd` `sub EDE , Rd`	3 cycles 3 cycles

Rotate	`rra Rs`
	`rra @Rs`
	`rra #3`
	`rra #4`
		1 cycle
		3 cycles
		3 cycles
		1 cycle

Multiply	`mov.w #9,&MPY`
	`mov.w #4,&OP2`
	`mov.w &RESLO,R5`
		4 cycles
		3 cycles
		3 cycles

ANSWER: （兩者狀況需全寫，只寫其中一種狀況者給 3 分）

[凱仁] 因執行 multiply 所需的 cycle 數固定，約 10 個 cycle，當數字較小時，`mov` 及 `rra` 指令所需數量也較少，所以軟體執行速度較快；但當數字越來越大時，需要 rotate 與加法的數量也會增加，相對硬體乘法器仍然只需 10 個 cycle，所以當數字變大時，硬體乘法器效能較好。

8. **[3%] [Instructions]**

How to implement logical OR operation between two numbers by MSP430?

ANSWER: [呂源] `bis Rx, Rn ; to do Rn = Rx | Rn`

9. **[Instructions]**

(a) **[3%]** What is the machine code of "`mov.b #00000100b, &p2out`" for MSP430F2013?

ANSWER: [凱仁] "`42E2 0029`"（第一個 word 2 分，第二個 word 1 分）

(b) **[3%]** The machine code is `40F2 0040 0029`

Is the low active LED connected to P2.6 ON or OFF? Why?

ANSWER: [凱仁] LED OFF（答案 1 分；原因 2 分）

The Assembly code is "`mov.b #01000000b, &p2out`".

opcode				source	Ad	B/W	As	destination	Two-operand arithmetic
0	1	0	0	source	Ad	B/W	As	destination	**MOV** Move source to destination
0	1	0	1	source	Ad	B/W	As	destination	**ADD** Add source to destination
0	1	1	0	source	Ad	B/W	As	destination	**ADDC** Add source and carry to destination
0	1	1	1	source	Ad	B/W	As	destination	**SUBC** Subtract source from destination (with carry)
1	0	0	0	source	Ad	B/W	As	destination	**SUB** Subtract source from destination

10. **[6%] [Function / Interrupt]**

Please briefly describe the behavior when an interrupt occurs and exits.

ANSWER: [呂源]（進去 5 分、出來 1 分；原則上以下缺一項各扣 1 分，扣至本題 0 分為止）

Occurs:

①. Any currently executing instruction is completed.

②. The PC, which points to the next instruction, is pushed onto the stack.

③. The SR is pushed onto the stack.

④. The interrupt with the highest priority is selected if multiple interrupts occurred during the last instruction and are pending for service.

⑤. The interrupt request flag resets automatically on single-source flags. Multiple source flags remain set for servicing by software.

⑥. The SR is cleared. This terminates any low-power mode. Because the GIE bit is cleared, further interrupts are disabled.

⑦. The content of the interrupt vector is loaded into the PC; the program continues with the interrupt service routine at that address.

Exits:

①. The SR with all previous settings pops from the stack. All previous settings of GIE, CPUOFF, etc. are now in effect, regardless of the settings used during the interrupt service routine.

②. The PC pops from the stack and begins execution at the point where it was interrupted.

11. **[6%] [Coding]**

The values of a set of data lie between 0.12 and 1.5. Please describe how you calculate with those data by MSP430?

ANSWER: [呂源] 同乘以 100 或其他數字，讓數字變成整數，CPU 的 ALU 才能計算，最後計算結果再將常數除回來。（提及變成整數，或轉成「IEEE floating point」處理給 6 分，其餘原則上沒分）

12. **[10%] [Function / Interrupt]**

A student uses `call #DiscreteFourier` in `PORT1_ISR` routine. Please briefly describe what's wrong with the implementation and rewrite the appropriate Assembly code. (Only necessary code is required.)

ANSWER: [呂源] ISR 不應該被放入一些 heavy task。（敘述理由 4 分，code 6 分，錯一項扣 1 分）

```
//set somewhere to be a signal let infinite loop to rolling it then do it
while(1){
    if(Do_Fourier == 1)
        call #DiscreteFourier
}

PORT1_ISR:
    ...
    Do_Fourier = 1
    ...
```

13. **[Coding]**

```
#include "msp430.h"
       ORG    0FFFEh
       DC16   init
       RSEG   CSTACK
       RSEG   CODE
TEST   MACRO x,y
LOCAL  loop,out
       mov x,R5
       mov y,R6
loop:  cmp R5,R6
       jn out
       add #1,R5
       jmp loop
out:   add #2, R6
       ENDM

init:  mov    #SFE(CSTACK), SP
main:
       TEST #6,#12
       sub #1,R6
       TEST #3,#7

       jmp $
       END
```

(a) **[3%]** How many times does the loop execute?

ANSWER: [凱仁] 14

(b) **[3%]** What are the final values of R5 and R6?

ANSWER: [凱仁] R5 = 8, R6 = 9 （錯一項扣 1 分）

(c) **[2%]** What kind of control operation does the MACRO TEST do? (e.g.: for loop …)

ANSWER: [凱仁] while

14. **[9%] [Coding]** Please find and describe three errors in the following Assembly code and then correct it.

```
#include "msp430.h"
haha MACRO x
mylabel:
       ;do something
       ENDM
       NAME    main
       PUBLIC  main

       ORG     0FFF8h
       DC16    init

       RSEG    CSTACK
       RSEG    CODE
init:  mov     #SFE(CSTACK), SP

teststring:
       DW "Wrong"
       DW "\0"

main:  nop
       mov.w   #WDTPW+WDTHOLD,&WDTCTL
       haha 1
       haha 2
       ;do something
       jmp $

       END
```

ANSWER: [祐農]（每項各 3 分：找到錯誤 2 分，更正程式碼 1 分）

①. init vector is `0fffeh`

②. `teststring` 定義的是資料，但放在這裡，在程式執行時會被視為是指令，而造成錯誤。

③. Macro's local label 需要宣告 local。

15. **[Coding]** A C function is compiled to be an Assembly subroutine. Draw the **stack frame** and show the appropriate contents and pointer of the stack at each of the following stage and write the corresponding Assembly code for accessing the stack, while

(a) **[5%]** Passing the function parameters to subroutine,

(b) **[5%]** Preserving the used registers in subroutine,

(c) **[5%]** Assigning the local variables in subroutine

(d) **[8%]** How is the stack pointer restored to its original SP value before the C function is executed?

The followings show the parameters, local variables, and used registers in this question.

i. Two parameters of the C function are passed by stack.

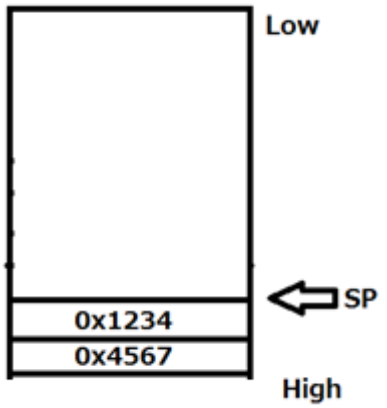
ii. There are two local variables in the following types.
`int`, `unsigned long long int`

iii. R6, R7 and R8 are used in the subroutine.

ANSWER: [呂源]

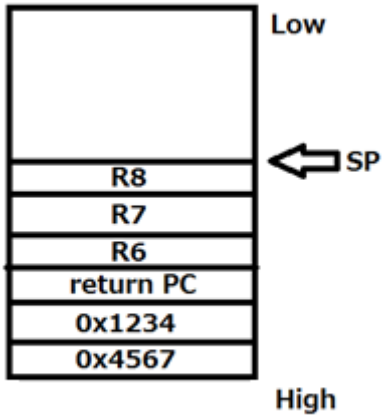
(a)（Coding 3 分，Stack 圖表 2 分）

```
;if push 0x4567
mov #0x4567 , r15
push r15

;if push 0x1234
mov #0x1234 , R12
push R12
```



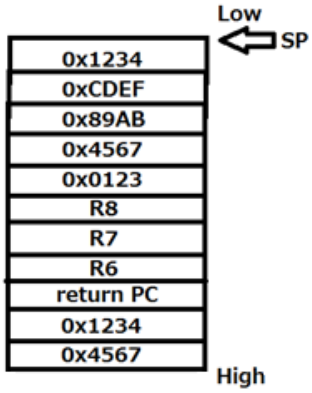
(b)（Push 一個 1 分，Stack 圖表 2 分）

```
push R6
push R7
push R8

;do something

;pop R8
;pop R7
;pop R6
```



(c)（sub 指令與長度 1 分，Stack 順序 2 分；Stack 圖表根據長度，對 2 分，錯 1 分）

```
;Two local variables example:
;local a = 0x0123 4567 89AB CDEF
;local b = 0x1234
sub #10, SP
mov 0x1234,0(SP)
mov 0xCDEF,2(SP)
mov 0x89AB,4(SP)
mov 0x4567,6(SP)
mov 0x0123,8(SP)
```



(d) 有幾個參數等 return 的時候再 pop 出來。（Coding 4 分，Stack 圖表 4 分）

```
;當完成 subroutine 的時候
;希望將 stack 使用狀態恢復至使用 subroutine 之前
;1.先將 local variables 使用的空間恢復

add #10, sp

;2.將 saved register 還原到原本的位置

pop  r8
pop  r7
pop  r6

;3.將 return pc 從 stack 放回 pc

reta                ;ret 與 reta 皆可

;4.(實際上 C call asm library 的時候
;    reta 之後會自動將參數拿掉
;    (IAR 的 disemble 也沒有看到)
;    所以以下範例假設為 asm call asm)
;;傳入參數
;;call
;;退出參數

add 4, SP
```