微處理機系統

作業一:　Andes N801-S Assembly Programming

資工三乙 406262084 梁博全
資工三乙 406262319 黃育晧

繳交日期
— Wen, Sep 25, 2019

Part 1: General Problem

Q1: Show the basic architecture of a computer system.

- Input Unit
  - 向資訊處理裝置(電腦手機)提供資料和訊號
  - 鍵盤、滑鼠、觸控筆等等
- Output Unit
  - 從資料處理系統接受資料和指令並執行特定任務
  - 螢幕，投影機等等
- Storage Unit
  - 可以載入、保存以及讀取資料的設備。
- Arithmetic Logic Unit
  - 主要功能是進行二進位的算術運算
- Control Unit
  - 有時為CPU一部分，透過該裝置的運行來控制其他裝置的活動

Q2 Show the difference between microprocessors(微處理器)and microcontrollers(微控器).

- 微控器
  - 將CPU、RAM、ROM、I/O 或A/D等周邊關於記憶與運算功能整合在一起
- 微處理器
  - 由一片或少數幾片大規模集成電路組成的中央處理器。這些電路執行控制部件和算術邏輯部件的功能。

- 差別
  - 微控制器完整結構除了CPU，還包刮RAM、ROM、計時器和接口等等。微處裡器則是一個單晶片CPU
  - 微處理器通常作為微型計算機系統中的CPU使用。微控制器通常用於面向控制的應用。
  - 微控制器價位相對較低。體積較小，功耗和成本下降、可靠性提高。
  - 微處理的指令集適合用來操作大規模的數據指令，而微控制器則是用於輸入/輸出

Q3 What are the addressing modes (定址模式) while writing assembly programs?

- Indirect Addressing 間接定址
- Register Addressing 暫存器定址
- Register Indirect Addressing 暫存間接器定址
- Displacement Addressing 位移定址
- Relative Addressing 相對定址
- Base-Register Addressing 基底暫存器定址
- Indexed Addressing 索引定址
- Stack Addressing 堆疊定址

Q4 how the types of buses?

- ISA - Industry Standard Architecture
- EISA - Extended Industry Standard Architecture
- MCA - Micro Channel Architecture
- VESA - Video Electronics Standards Association
- PCI - Peripheral Component Interconnect
- PCI Express (PCI-X)
- PCMCIA - Personal Computer Memory Card * Industry Association (Also called PC bus)
- AGP - Accelerated Graphics Port
- SCSI - Small Computer Systems Interface.

Part 2: Assembly Programming

hello.s

```
.section .rodata
.align 2
.LC0:
    .string "Andes Assembly Programming\n"
    .text
    .align 2
    .global main

main:
.LFB2:
    movi  $r6, 0
.L2:
    move  $r7, $r6
    sltsi $r7, $r6, 5
    beqz  $r7, .L3
    la    $r0, .LC0
    bal   printf
    addi  $r6, $r6, 1
    b .L2
.L3:
    ret
```

Q1 詳細說明機器基本資料型別及其儲存空間大小

Data Type	**Size**
Bit	1 bit
Character	8 bits
Byte	8 bits
Halfword	16 bits
Word	32 bits
Integer	32 bits

Q2 16-位元及**32-**位元指令的編碼差異**?**

- CPU 在執行指令按照字節(8位)執行
- 16位元的指令呼叫CPU在一個週期內執行2個字節
- 32位元的指令呼叫CPU在一個週期內執行4個字節

Q3 詳細說明呼叫副程式時，如何傳遞參數及如何回傳 **return value**

- 先將參數 `push` 進去
- 再 `push` return address
- 將副程式的位置傳遞到暫存器

- 呼叫副程式，並執行
- 執行完成後， `pop` 出回傳值
- `pop` return address 以回到主程式

Q4 找出此程式中之假指令，說明其功用為何?

- `.section`
  - 以下區塊為user defined section
- `.rodata`
  - 唯獨資料
- `.align`
  - 設定間距
- `.string`
  - 設定字串內容
- `.text`
  - 定義下面區塊為程式碼
- `.global main`
  - exporting main
- `la $r0, .LC0`
  - 將 `.LC0` 的位置放到 `$r0`
- `bal printf`
  - 呼叫printf函數

pseudo.s

```
.section .rodata
.align 2
err_msg0:
    .string "Error Happens in Subroutine 0\n"
err_msg1:
    .string "Error Happens in Subroutine 1\n"
err_msg2:
    .string "Error Happens in Subroutine 2\n"
err_msg3:
    .string "Error Happens in Subroutine 3\n"
    .data
    .align 2
var1:
    .word -42
var2:
    .word 42
numlist:
    .byte 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88
    .text
    .global main

main:
    nop
    bal  sub0
    bal  sub1
    call sub2
    la   $r6, sub3
    bral $r6
exit:
    ret

sub0:
    move $r6, 0x55aa
    slti $r6, $r6, 0x3fff
    blez $r6, 1f
    la   $r0, err_msg0

    push $lp
    bal printf
    pop $lp

1:
    ret

sub1:
    li    $r6, #42
    la    $r7, var1
    lw    $r8, [$r7]
    add   $r8, $r8, $r6
    beqz  $r8, lf
    la    $r0, err_msg1

    push $lp
    bal printf
```

```
55      pop $lp
56
57  1:
58      ret
59
60  sub2:
61      move    $r6, #0x12345678
62      l.w     $r7, num1
63      neg     $r7, $r7
64      add     $r7, $r7, $r6
65      beqz    $r1, 1f
66      la      $r0, err_msg2
67
68      push    $lp
69      bal     printf
70      pop     $lp
71
72  1:
73      ret
74
75      .align 2
76  num1:
77      .word   0x12345678
78
79  sub3:
80      push.w var1
81      pop $r6
82      l.w $r7, var2
83
84      add  $r7, $r7, $r6
85      beqz $r7, 123f
86      la   $r0, err_msg3
87
88      push $lp
89      bal printf
90      pop $lp
91
92  123:
93      ret
```

Q1 何謂假指令?

- 假指令並不是硬體上的指令，是為了程式設計師方便撰寫而有的指令。它指示組合語言如何編譯程式的一種指令。

Q2 找出此程式中所有假指令，並說明每個假指令之功用為何與其對應之實際機器指令?

- `.section`
    - 以下區塊為user defined section
- `.rodata`

    - 唯獨資料
- `.align`
    - 設定間距
- `.string`
    - 設定字串內容
- `.data`
    - data section
- `.text`
    - 定義下面區塊為程式碼
- `.global main`
    - exporting main
- `bal` (line 24)
    - 跳到sub
    - depending on how it is assembled
- `call` (line 26)
    - 呼叫
- `la`
    - 把address放到暫存器
- `bral`
    - 呼叫暫存器
- `move`
    - 將一暫存器內的值複製一份並移動到另一個暫存器內
- `push`
    - 將暫存器放入stack
- `bal printf`
    - 呼叫printf函式

- pop
  - 將stack內暫存器取出
- neg
  - 取2補數

blockcopy.s

```
.equ num,20
.text
.global memcpy
.global main

main:

memcpy:
    la $r1, src
    la $r0, dst
    li $r2, num

m_word_copy:
    lmw.bim $r5, [$r1], $r8, 0
    smw.bim $r5, [$r0], $r8, 0
    addi    $r2, $r2, #-4
    bnez    $r2, m_word_copy
    ret

    .data
    .align 2

src:    .word  1,  2,  3,  4,  5,  6,  7,  8,  9, 10
        .word 11, 12, 13, 14, 15, 16, 17, 18, 19, 10

dst:    .skip num*4
        .end
```

Q1 解釋 lmw.bim 與 smw.bim 指令之功用為何?

- `lmw.{b,a}{i,d}{m?} Rb, [Ra], Re, Enable4`
- `smw.{b,a}{i,d}{m?} Rb, [Ra], Re, Enable4`
  - `{b,a}` ：在做load或store之前或之後(before/after)改變記憶體 `Ra` 的位置
  - `{i,d}` ：遞增或遞減(increasing/decreasing)的去改變記憶體 `Ra` 的位置
  - `m` ：是否立即更新記憶體 `Ra` 的位置
  - `Enable4` ：為一個4bits的數字，用以表示 `$r28 ~ $r31` 是否參與本次操作

Q2 說明此程式目的為何?

- 將src陣列中的內容透過 `lmw` ， `smw` 複製到dst陣列

macro.s

```
.macro load_imm32 rt5, imm32

.if ( ( \imm32 <= 0x7ffff ) && ( \imm32 >= -0x8000 ) )
    movi \rt5, \imm32
.elseif ( \imm32 & 0x00000fff == 0x0 )
    sethi \rt5, hi20(\imm32)
.else
    sethi \rt5, hi20(\imm32)
    ori \rt5, \rt5, lol2(\imm32)
.endif
.endm
.global main
.text

main:
! both sethi and ori
    load_imm32 $r2, #0x12345678
! sethi only
    load_imm32 $r3, -0x12345000
! movi only
    load_imm32 $r0, #0xfff

end:
    ret
    .end
```

Q1 說明macro用途?

- 方便工程師重構想同的邏輯，把程式邏輯相同的部分，撰寫成macro，將動作化成指令。
- macro 的效能比function好，因為不需要function call

Q2 Assembler如何處理macro?

- 定義好macro內容及名字後，當在編譯時，macro內容會取代macro名稱，每次開會在複製一次內容。

質數產生C程式&組語程式(輸出小於100的質數)

- a.C程式碼(不須行號)，不准用圖檔。演算法說明。
- 說明
    - 用一陣列(prime)表示其index的數字是否為質數，如果是 `prime[index] = 1` 反之則0
    - 先將陣列全部設成1

- 接著從i=2開始判斷，如果i是質數的話，將i的倍數全部設成0, 因為i的倍數一定有i這個因數，所以一定不是質數

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){

    int prime[100],i,j;

    for( i = 2 ; i < 100 ; i++ )
        prime[i] = 1;

    int k = 0;
    for( i = 2 ; i < 100 ; i++ ){
        if( prime[i] ){
            for( j = i+i ; j < 100 ; j+=i )
                prime[j] = 0;
            printf("%d ",i);
            k++;
            if( k%10 == 0 )
                printf("\n");
        }
    }

}
```

- b.組語程式碼(須有行號)，不准用圖檔。此程式中必須有註解說明對應到的C code。

```
.equ num ,100

.section .rodata
.align 2

prime:
    .space num*4                    ! int prime[100]

.str1:
    .string "%d "
    .text
.str2:
    .string "\n"
    .text
    .global main

main:
    movi   $r9, 1                   ! r9 = i = 1
    la     $r8, prime               ! r8 = prime address
    movi  $r10, 1

.set_prime:

    sw    $r10, [$r8+($r9<<2)]      ! prime[i] = 1
    addi   $r9, $r9, 1              ! i++
    move   $r7, $r9
    sltsi  $r7, $r7, num            ! i < 100
    bnez   $r7, .set_prime

    ! end set_prime

    movi   $r9, 2                   ! r9 = i = 2
    movi   $r4, 1                   ! r4 = 1
    movi   $r6, 0                   ! int k = 0

.Loop1:
    move   $r7, $r9
    sltsi  $r7, $r7, num            ! i < 100
    beqz   $r7, .finish
    lw    $r10, [$r8+($r9<<2)]      ! r10 = prime[i]
    movi   $r4, 1
    beq   $r10, $r4, .Loop2         ! if( prime[i] )
    addi   $r9, $r9, 1              ! i++
    b         .Loop1

.Loop2:
    move   $r3, $r9                 ! int j = i
    add    $r3, $r3, $r9            ! j = j + i
    movi   $r5, 0

.sieve:
    move   $r7, $r3
    sltsi  $r7, $r7, num            ! j < 100
    beqz   $r7, .print_prime
```

```
55      sw     $r5, [$r8+($r3<<2)]      ! prime[j] = 0
56      add    $r3, $r3, $r9            ! j = j + i
57      b   .sieve
58
59  .print_prime:
60      move   $r1, $r9
61      la     $r0, .str1
62      bal printf                      ! printf("%d ",i)
63      addi   $r9, $r9, 1              ! i++
64      addi   $r6, $r6, 1              ! k++
65      movi   $r4, 10
66      divsr  $r1, $r2, $r6, $r4       ! if( k%10 == 0 )
67      beqz   $r2, .print_enter
68      b   .Loop1
69
70  .print_enter:
71      la     $r0, .str2
72      bal printf                      ! printf("\n")
73      b   .Loop1
74
75  .finish:
76      ret
77      .end
```