uOttawa

L'Université canadienne
Canada's university

CEG3136
Computer Architecture II

Midterm Exam Review

Midterm Exam Review

- Midterm Exam is a close-book examination;
- Calculators are allowed;
- Time: 80 minutes

CPU and Memory for Part 1

	A	B
D	60	00
X	2004	
Y	4470	
PC	2012	
SP	200C	

	S	X	H	I	N	Z	V	C
CCR	1	0	0	1	0	1	0	1

Microprocessor

$0000	
$2000	F5
	02
	10
	25
	09
	09
	85
	77
$2008	5D
	47
	C1
	61
	FC
	DD
	99
	E3
$2010	B3
$FFFF	

Memory

All values shown are hexadecimal.

Part 1a – Short Answer Questions

1) Consider the following instructions:

LDAA #56

LDAB $2002

ABA

What will be the contents of Accumulator A after the execution of ABA? ____________

- How will the ABA instruction affect the condition codes NZVC? ___________

Part 1a – Solution:

1) Consider the following instructions:

LDAA #56 => A = $38

LDAB $2002 => B = $10

ABA => A = $48

What will be the contents of Accumulator A after the execution of ABA? ___48________

- How will the ABA instruction affect the condition codes NZVC? _0000_________

Part 1a – Short Answer Questions

2) Give the contents of D register after the execution of the following 2 instructions. ________

LDAA 1, +X

LDAB 1, X

Part 1a – Solutions:

2) Give the contents of D register after the execution of the following 2 instructions. __0985______

LDAA 1, **+**X X=X+1=2005,

(2005)->A => A=09

LDAB 1, X (X+1) -> B => B=85

Part 1a – Short Answer Questions

3) Will the branch in the following instructions execute (i.e. will the program branch)? Give the contents of Accumulator A after the execution of the TSTA instruction.

```
VAR3    EQU   $200E
        LDAA  VAR3
        TSTA
        BLT   NEXT
.
.
NEXT    LDAB  VAR3
```

Branch (Y or N) _______? Contents of the A register ______?

Part 1a – Solution:

3) Will the branch in the following instructions execute (i.e. will the program branch)? Give the contents of Accumulator A after the execution of the TSTA instruction.

```
VAR3    EQU   $200E
        LDAA  VAR3        => A = 99
        TSTA              => A - 0
        BLT   NEXT        => branch to next
.
.
NEXT    LDAB  VAR3        => B = 99
```

Branch (Y or N) ____Y_____? Contents of the A register __99_____?

Part 1a – Short Answer Questions

4) Describe the changes made to memory by the instruction PSHA.

Part 1a – Solution:

4) Describe the changes made to memory by the instruction PSHA.

Before

=> SP = 200C

=> A = 60

After

PSHA

=> SP= SP - 1 = 200B

=> (200B) = 60

Part 1a – Short Answer Questions

5) Consider the following instructions

```
LEAS 3,X
PULD
STD -8, SP
```

The last instruction stores the contents of D into a location in memory. Give the address of this location ________________ and the value that gets stored there ______________

Part 1a – Solution:

5) Consider the following instructions

LEAS 3,X SP=X+3 = 2007

PULD => (2007:2008) -> (A:B)

775D -> D

=> SP+2 -> SP

SP = 2009

STD -8, SP => D ->(2001:2002)= 77:5D

The last instruction stores the contents of D into a location in memory. Give the address of this location (2001:2002) and the value that gets stored there __77:5D________

Part 1b – Short Answer Questions

(10 Points) Translate the following short C program into assembler. Use the stack to exchange ALL parameters and the result (assume *int*'s take 2 bytes of storage, and *byte*'s take 1 byte of storage). Ensure that the registers used by the subroutine are not changed after the subroutine has executed. Define the stack usage using the OFFSET directive and labels as offsets into the stack.

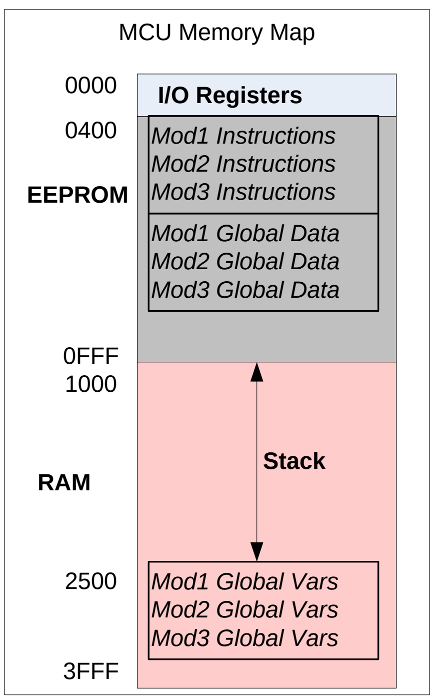
```
/*------------------------------
Function: addInts
Description: Adds two 8-bit integers.
------------------------------*/
int addInts(byte val1, byte val2)
{
    int sum;
    sum = val1 + val2;
    return(sum);
}
```

```
;-------------Assembler Code-------------
; Subroutine - addInts
; Parameters - val1 - on stack
; val2 - on stack
; Results sum - on stack
; Description: Adds two integers.
; Stack usage:
          OFFSET 0
          DS.W 1 ; preserve Register D
          DS.W 1 ; return address
ADI_SUM    DS.W 1 ; sum and return value
ADI_VAL1   DS.B 1 ; byte val1
ADI_VAL2   DS.B 1 ; byte val2
addInts:   PSHD ; preserve acc D
          CLRA
          LDAB ADI_VAL1,SP
          ADDB ADI_VAL2,SP
          ADCA #00
          STD ADI_SUM,SP
          PULD
          RTS
```

Part 2 Theory

(10 points) Modular design provides the means of separating the tasks of any project into manageable pieces. Each of the modules in a project is typically stored in a separate assembler file. Although this does help facilitate the development of a software project, it produces a challenge – how to collect the executable code, constant global data and variable global data from each module into separate sections of software where code and constant data are stored in Read Only Memory (these sections follow one another) and the variable global data is stored in RAM.

Describe how assembler directives, such as ORG, SECTION, SWITCH available in MiniIDE, can be used to perform this organization of software sections.



Module 1 (module1.asm)

```
code      SECTION
          ORG $0400
 ;---Assembler Instructions

dataEE    SECTION
          ORG ENDCODE
 ;---Constant global data

dataRAM SECTION
          ORG $2500
 ;---Global variables (modifiable)

;--------Add other Modules
  INCLUDE module2.asm
  INCLUDE module3.asm
;--Defines start of dataEE section
 SWITCH code
ENDCODE
```

Module 2 (module2.asm)

```
    SWITCH code
;---Assembler Instructions

    SWITCH dataEE
;---Constant global data

    SWITCH dataRAM
;---Global variables (modifiable)
```

Module 3 (module3.asm)

```
    SWITCH code
;---Assembler Instructions

    SWITCH dataEE
;---Constant global data

    SWITCH dataRAM
;---Global variables (modifiable)
```

uOttawa

Part 2 Theory: Solution

Directive SECTION – defines the start of a section.

Directive ORG – sets the location counter of a section.

The above two directives can be used to define sections and their locations in memory.

Directive SWITCH – changes section, such that any subsequent assembly instructions (including storage instruction such as DS) are placed in the section.

Note: Absolute addresses are defined only on the second pass and thus ENDCODE which defines the location of the global data is placed after all code assembly and consequently places the global constant data section after the code section.

- ENDCODE value can change when code changes when source code is re-assembled.

Part 3 – Application Question

The C standard library provides a function to compare two strings such as:

short **strcmp(char *str1, char *str2)**

A *string* of characters terminated with a null character is stored in the memory starting at address found in the pointer variable *str1* and a second string at address found in the pointer variable *str2*. Develop a structured assembly subroutine that compares the string pointed to by *str1* to the string pointed to by string *str2*. The subroutine returns:

- a positive value if the string referenced by str1 is alphabetically larger than the string referenced by str2,
- 0 if both strings are the same, and
- a negative if the string is referenced by str1 alphabetically smaller than the string reference by str2.

Assume single byte ASCII characters.

1. First provide a <u>C function </u>that illustrates the design of the subroutine.
2. Then translate the C function to <u>assembler code to subroutine</u>. Do not forget to comment your code.

Part 3 – Application Question

Hints:

- **Compare strings character by character.**
- **Strings are equal when the end of both strings is reached at the same time.**
- **As soon as a difference between 2 characters is encountered, strings are not equal, and the return value can be obtained by subtracting the character of the second string (reference by str2) from the character of the first string (referenced by str1). For example if str1 points to the string “abcde” and str2 points to the string “abedc”, then the return value shall be: ‘c’ – ‘e’ = -2.**
- **The type short is a one byte signed integer.**

Part 3 – Application Question

```
short strcmp(char *str1, char *str2)
{









}
```

Part 3 – Application Question Solution

```
short strcmp(char *str1, char *str2)
{
    // use str1 and str2 pointers
    short retval = 0;
    while(*str1 != '\0' || *str2 != '\0') //loop until end of equal strings
    {
        if(*str1 != *str2)
        {
            retval = *str1 - *str2;
            break;
        }
        else
        {
            str1++;
            str2++;
        }
    }
    return(retval);
}
```

Part 3 – Application Question Solution

Assembler Source Code:

```
; Subroutine: short strcmp(char *str1, char *str2)
; Parameters
;       str1 - address of first string - on stack and register x
;       str2 - address of second string - on stack and register y
; Returns in register B
;       -ve value: str1 less than str2
;       +ve value: str1 greater than str2
;       0: strings are the same.
; Local Variables
;       retval - return value in register B
; Description: Compares the strings and returns a value that
;reflects the results of the comparison.
;
; Stack Usage:
```

Part 3 – Application Question Solution

```
OFFSET 0
SCMP_RETVAL DS.B 1 ; return value
SCMP_PRY DS.W 1 ; preserve Y
SCMP_PRX DS.W 1 ; preserve X
SCMP_RA DS.W 1 ; return address
SCMP_STR1 DS.W 1 ; address of first string
SCMP_STR2 DS.W 1 ; address of second string
```

Part 3 – Application Question Solution

```
strcmp: pshx ; preserve registers
        pshy
        leas -SCMP_RETVAL,sp
        clr SCMP_RETVAL,sp
        ldx SCMP_STR1,sp ; get address of first string
        ldy SCMP_STR2,sp ; get address of second string
        clra
scmp_while: ; while(*str1 != 0 || *str2 != 0)
        tst 0,x
        bne scmp_if
        tst 0,y
        beq scmp_endwhile
scmp_if:
        ldab 0,x ; if(*str1 != *str2)
        cmpb 0,y
        beq scmp_else
        subb 0,y
        stab SCMP_RETVAL,SP
        bra scmp_endwhile
```

Part 3 – Application Question Solution

```
scmp_else
      inx
      iny
scmp_endif
      bra scmp_while
scmp_endwhile:
      ldab SCMP_RETVAL,SP
      leas SCMP_RETVAL,SP ; restore stack pointer
      puly ; restore registers
      pulx
      rts
```