Université d'Ottawa · University of Ottawa

École d'ingénierie et de technologie de l'information

COURS: CEG3136/CEG3536	**PROFESSEURS:** Voicu Groza Mohamed Ibrahim
SEMESTRE: Automne 2016	**DATE:** 1ier novembre, 2016
	HEURES: 19h – 20h30

EXAMEN DE MI-SESSION

NOM ET NUMÉRO D'ÉTUDIANT:_____/_____

Instructions:

- Répondez à toutes les questions sur le questionnaire. Cet examen se fait à livre fermé.
- Utilisez l'espace trouvée pour répondre aux questions suivantes. Si plus d'espace est nécessaire, utilisez l'arrière des pages.
- Montrez tout votre travail et calculs pour obtenir tous vos points.
- Les calculatrices sont permises.
- Lisez attentivement toutes les questions avant de débuter.
- L'annexe de l'examen vous fournit un tableau d'instruction CPU12, un modèle de référence et des pages brouillon.

1. Cet examen comprend 3 parties.

Partie 1	**Réponses courtes**	**22 points**	
Partie 2	**Théorie**	**8 points**	
Partie 3	**Application**	**20 points**	
Total		**50 points**	

Partie 1a – Questions à réponses courtes (2 points par question – total 12 points)

Répondez les questions 1 à 5 en utilisant le contenu de la mémoire et de l'UCT indiqué dans l'annexe. Le diagramme montre les conditions initiales pour chaque question.

1) En utilisant l'instruction de l'adressage indexé indirecte, trouvez la valeur contenue dans le registre D:

LDD [4, Y]

Solution D = ((Y + 4)) = (0900**)** = F502 ➔ **D =** $F502

2) Identifiez la destination des données et la valeur des données de la dernière instruction dans la séquence suivante.

PULA
SUBA #1
STAA 4, SP

PULD - Loads FC into A, adds 1 to SP to give 090D
SUBD #1 Decrements contents of A by 1 => FB
STD 4, SP EA = 090D +4 = 0911

Destination: ***$0911***

Valeur des données : ***FB***

3) Examinez les instructions suivantes:

LDAB $1500
CMPB -4, X
BGT next
.
.
NEXT LDAA $0908

Est-ce que l'instruction BGT branchera à NEXT?

YES (B shall contain $4A and compared to $F5 at address $0900 ($0904 – 4), sinced the BGT is a signed compare and a +ve number > -ve number ($F5), then the branch will be taken).

Que sera le contenu du registre B après l'exécution de l'instruction CMPB? **__$4A_**

4) Quel sera le contenu du registre X après l'exécution des instructions suivantes?

LDY 0, X Y = (X) = ($0904) = $1503

LDX 3, Y X = (Y+3) = (1503 + 3) = (1506) = $**145D**

5) Quel sera le contenu de l'accumulateur B et les valeurs des bits d'états NZVC après l'exécution des instructions suivantes :

LDAB $0902
SUBB 4, SP B = ...**$0B**.....=> NZVC = ….**0000**.....

Subtracts $05 at 4,SP (090C+4 = 0910) from $10 at $0902 = $0B, no overflow (signed and unsigned, +ve result, non-zero result

6) Écrire les instructions nécessaires pour allumer et éteindre les 8 DELs (LEDs) du port A.

; Registre du port parallèle A
PORTA EQU $00 ; Registre de donnée du Port A
DDRA EQU $02 ; Registre de direction de donnée du Port A

```
movb   #$FF, DDRA ; Configurer comme sortie
movb   #$FF, PORTA ; allumer DELs
movb   #$00, PORTA ; Étendre DELs
```

Partie 1b – Questions à réponses courtes (total 10 points)

Translate the following C Pseudo-code to assembler.

C Function	Assembler Code

`//Global Variables/Constants`
`#define LENGTH 15`
`int array[LENGTH];`
`int *arrPt = array;`
 	
`/* ----------------------------`
`Function: addtoArray`
`Parameters: int val - a value`
`to add to the array.`
`Description: Stores 16-bit`
`value val in array using`
`global pointer arrPt.`
`arrPt is incremented so`
`that the next call will`
`add the value to the next`
`element in the array.`
`The pointer wraps to start`
`when it reaches the end of`
`the array.`
`----------------------------*/`
`void addToArray(int val)`
`{`
`*arrPt = val; // save value`
`arrPt++; // increment pointer`
`// Wrap around`
`if(arrPt == array + LENGTH)`
`arrPt = array;`
`}`
	`% GLOBAL Variables/Constants`
	`LENGTH EQU 15`
	`array DS.W LENGTH`
	`arrPt DC.W array`
	
	`%-----------------------------------`
	`Subroutine: addToArray()`
	`Parameters: val - in D register`
	`Returns nothing.`
	`Description:`
	`Stores 16-bit value (D) in array`
	`using pointer arrPt.`
	
	`%-----------------------------------%`
	`addToArray`
	`pshx`
	`ldx arrPt`
	`std 2,x+`
	`cpx #(array+2*LENGTH); % 2 bytes per int`
	`bne endif`
	`ldx #array`
	`endif:`
	`stx arrPt`
	`pulx`
	`rts`

Partie 2 Théorie (totale 8 points)

(10 points) Décrivez comment l'exécution des instructions CPU12 (tel que bsr, jsr, rts) permettent de faire appel à sous-programme. (Vous **n**'avez **pas** à décrire comment les valeurs sont échangées avec le sous-programme appelé) Donnez un exemple concret (avec de vraies adresses) d'un appel à un sous-programme comme illustration.

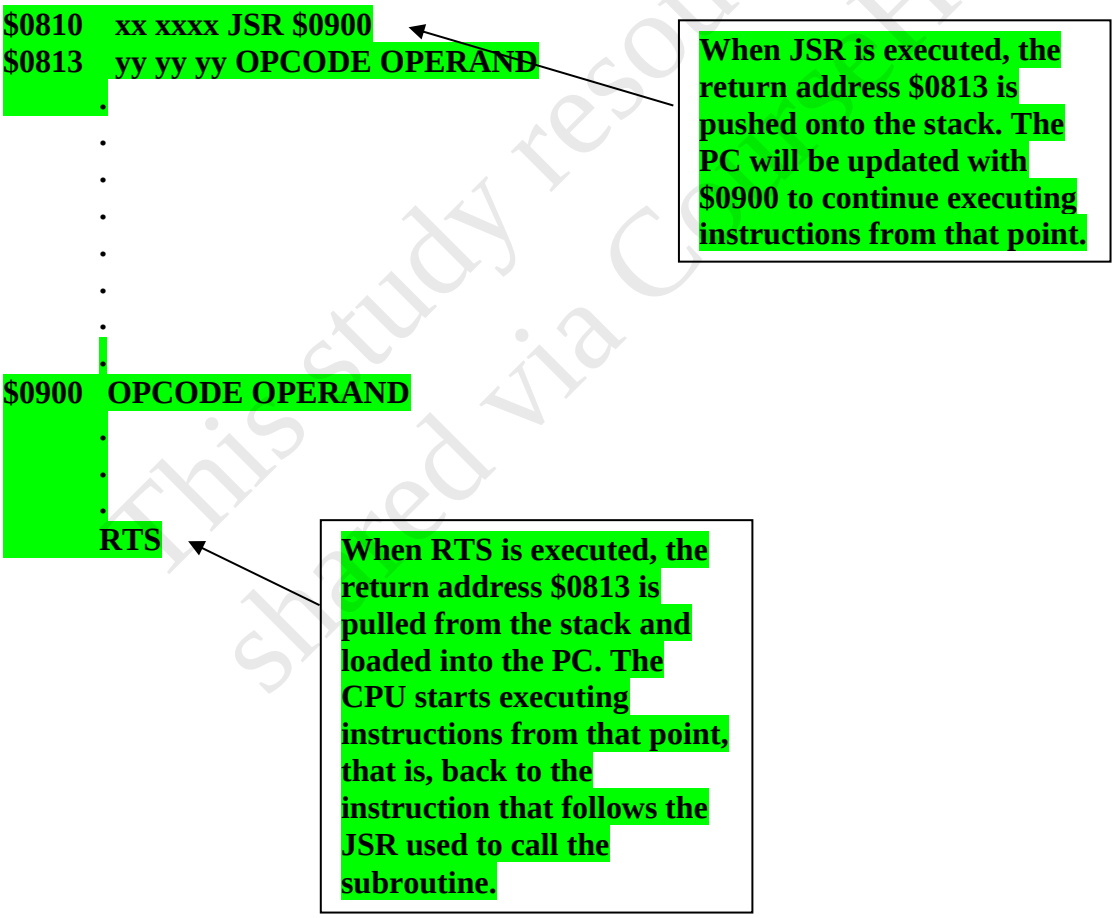
Description

The stack is used to store the return address so that the CPU can find the return address when it reaches the end of the sub-routine.

The return address is found in the PC when either of these instructions is executed, that is, the PC points to the instruction following the JSR or BSR instruction. Thus the content of the PC is pushed onto the stack when a JSR or BSR instruction is executed.

When the RTS at the end of the subroutine is encountered, the PC is changed by pulling a 16-bit value from the stack. Thus the execution will continue from the instruction following the JSR or BSR used to jump into the subroutine.

Exemple:



Partie 3 – Application (total 20 points)

La bibliothèque standard C offre une fonction pour comparer deux chaînes tel que :

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

Une *chaîne* de caractères terminée avec le caractère nul est stockée en mémoire à l'adresse retrouvée dans la variable pointeur *str1* et une deuxième chaîne se retrouve à l'adresse retrouvée dans la variable pointeur *str2*. Développez un sous-programme structuré qui compare la chaîne pointée par *str1* à la chaîne pointée par *str2*. La fonction retourne :

- 0 si les deux chaînes sont identiques,
- Une valeur positive si la chaîne référée par *str1* est alphabétiquement plus petite que la chaîne référée par *str2*,
- Une valeur négative si la chaîne référée par str1 est alphabétiquement plus grande que la chaîne référée par *str2*.

Prenez pour acquis que les caractères sont des caractères ASCII d'un octet.

1. Premièrement donnez une fonction C qui donne la conception de votre sous-programme.
2. Ensuite traduisez la fonction C à un sous-programme assembleur. N'oubliez pas de commenter votre code.

Indices:

- Comparez les chaînes caractère par caractère.
- Quand les caractères sont différents, les chaînes ne sont pas égales, et la valeur de retour est obtenue en soustrayant le caractère de la deuxième chaîne (référé par str2) du caractère de la première chaîne (référé par str1). Par exemple, si str1 pointe "abcde" et str2 pointe la chaîne "abedc", alors la valeur de retour pour le troisième caractère est: 'c' - 'e' = -2.
- Les chaînes sont identiques lorsque la fin de deux chaînes est atteinte en même temps.
- Le type `short` est un nombre entier signé d'un octet.

Conception (Fonction C):

```
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);
}
```

Code source assembleurr:

```
; 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
;       int  -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 velue that reflects
;              the results of the comparison.
;
; Stack Usage:
        OFFSET 0
SCMP_RETVAL  DS.B 1  ; return value
SCMP_VARSIZE
SCMP_PRY  DS.W 1  ; preserve B
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

strcmp: pshx    ; preserve registers
        pshy
        leas -SCMP_VARSIZE,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
scmp_else
       inx
       iny
scmp_endif
        bra scmp_while
scmp_endwhile:
        ldab SCMP_RETVAL,SP
        leas SCMP_VARSIZE,SP ; restore stack pointer
        puly    ; restore registers
        pulx
      rts
```