Student ID: _____

EDE1203

Data Structures and Algorithms

MOCK-Examination - Trimester 2, AY2020/21

Friday, 18/02/2021

02:30pm - 04:10pm

(90 minutes plus 10 minutes reading time)

Instructions to students:

1. This examination comprises **NINE (9)** sections and **EIGHTEEN (18)** questions printed on **FOURTEEN (14)** pages including one cover page and two pages for notes.

2. **ALL** answers to be written on the spaces provided. Additional answer sheets will not be accepted. You may use the last two pages in case you need additional space. Indicate at the provided question space if there is content on the last two pages that should be marked.

3. This is a **CLOSED BOOK** examination. Writing materials including drawing pens, pencils, crayons, compass, ruler, triangle, and circle template are allowed.

4. Laptops, calculators, mobile phones, smart watches, and wireless devices are **NOT** allowed in the examination hall. Please **switch off** your mobile phones during the examination.

5. You will be given 10 minutes of reading time. No writing is allowed during the reading period. You may only begin writing upon Chief Invigilator's instruction.

6. Do not detach any page of the exam.

7. Please use only **BLUE** or **BLACK** pens for writing.

At the end of the examination:

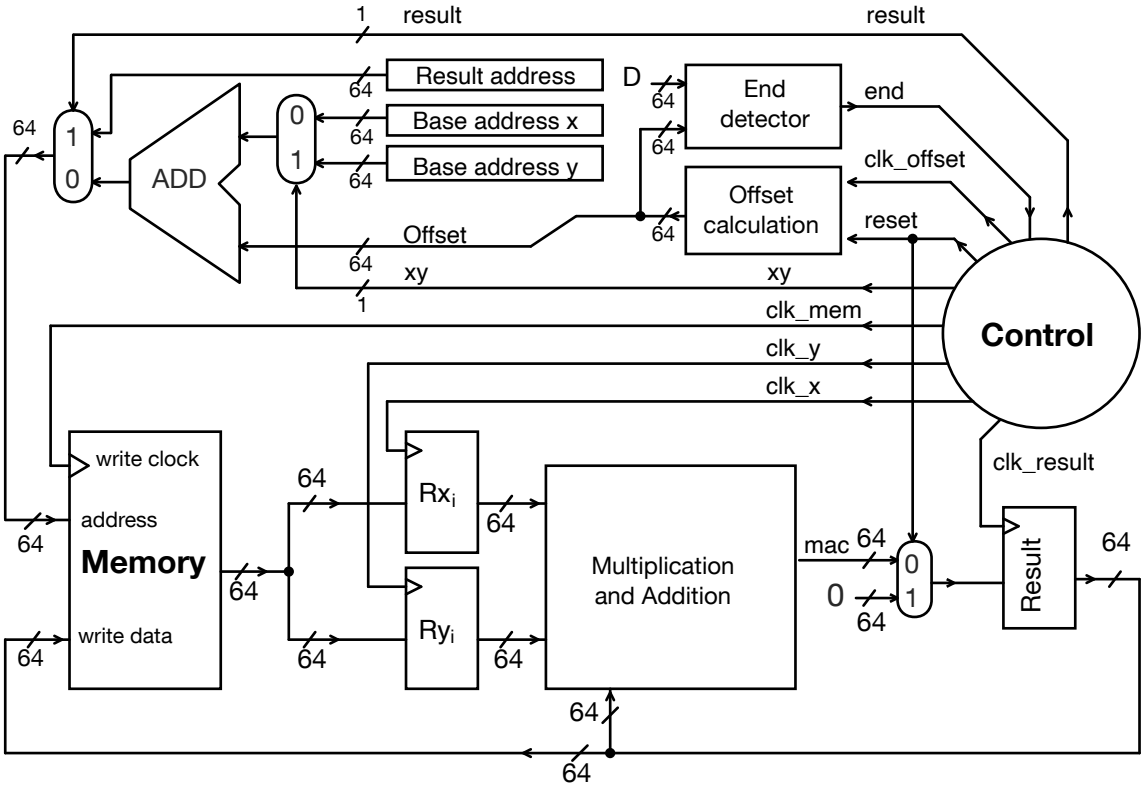
Please ensure that you have written you Student ID on each page on the top right corner. Failure to do so will mean that your work will not be identified.

The University reserves the right not to mark your script if you fail to follow these instructions.

Contents

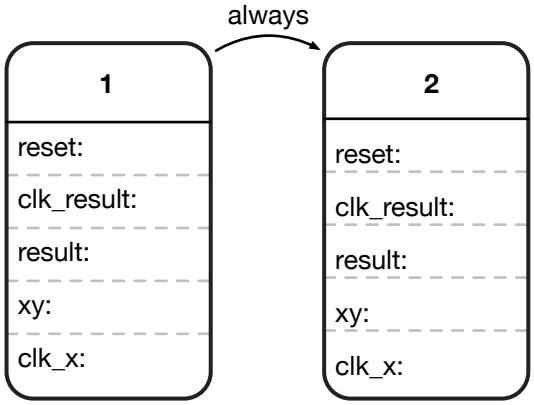
1 Datapath and Moore machine (5 marks)

See the following datapath.



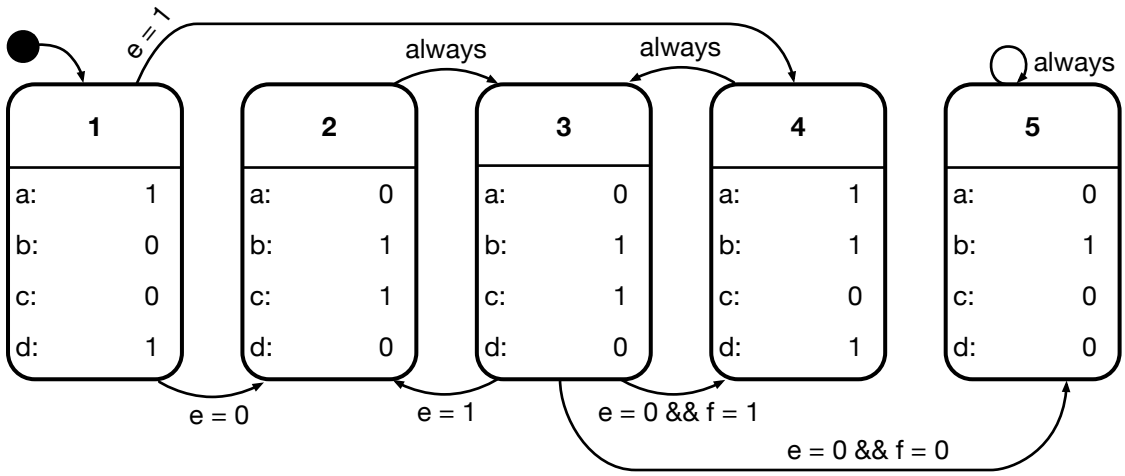
a) Provide to the following Moore machine (see question b) control signals that initialize register Result with 0 when going from state 1 to state 2. **(2 marks)**

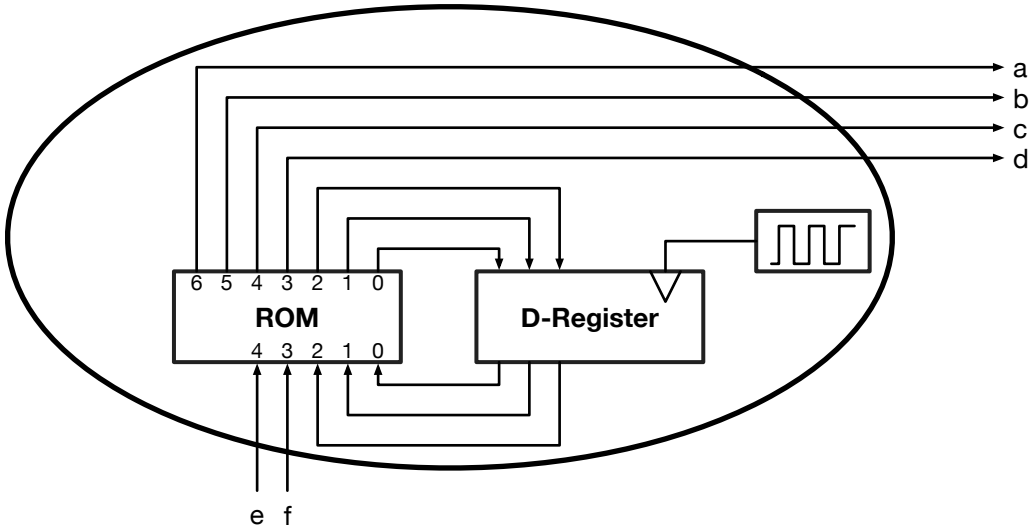
b) Provide to the following Moore machine control signals that – when going from state 1 to state 2 – write to register Rx_i the 64 bit data word that is stored at memory address calculated by *Offset* + *Base address x*. **(3 marks)**



2 ROM based control unit (5 marks)

The following Moore machine will be implemented with a ROM based circuit as follows.





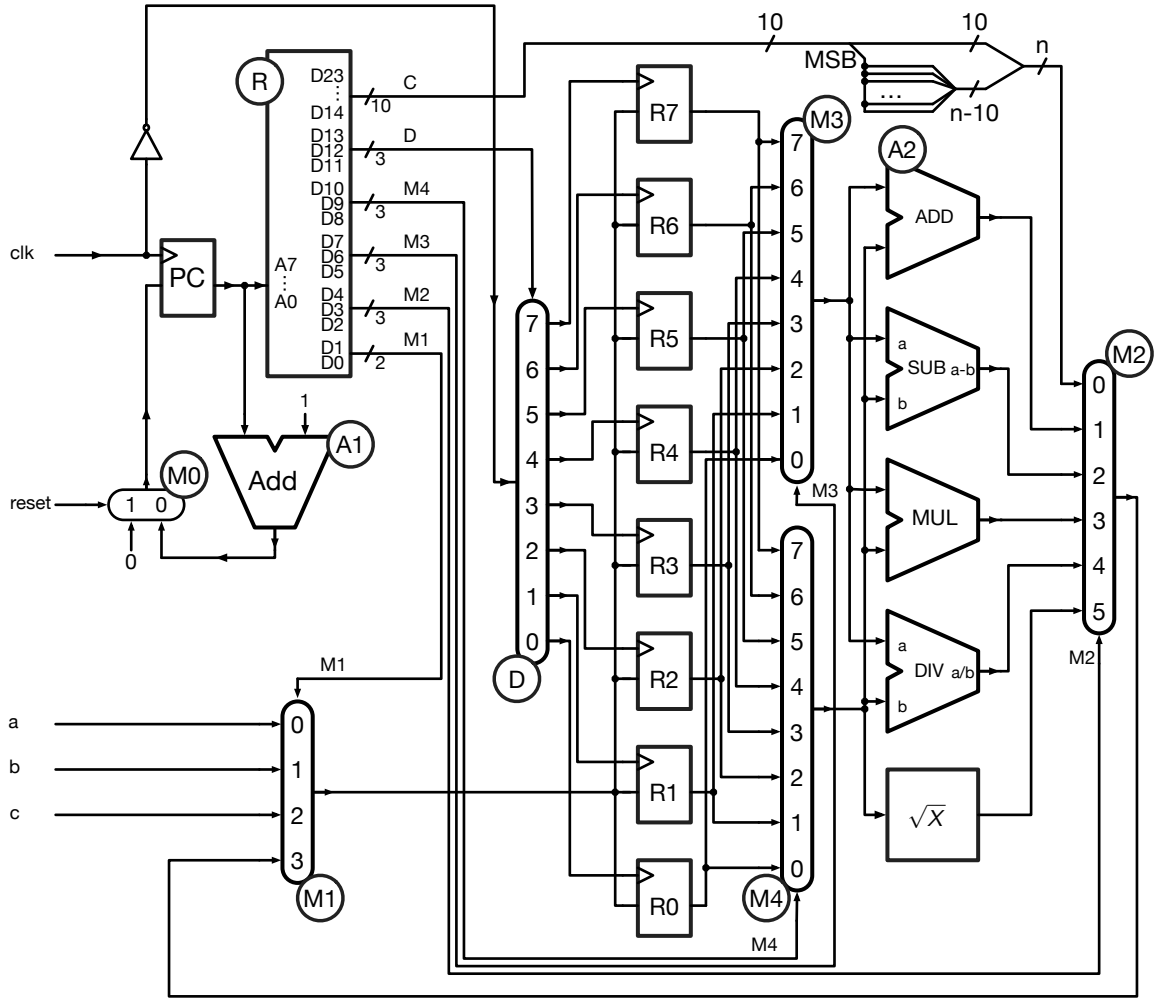
a) For the given addresses, provide in binary the contents of the ROM memory to implement the Moore machine. **(5 marks)**

e f
00 001
00 011
01 011
11 001
11 011

Adresse	Daten 6	5	4	3	2	1	0
0x01	1	0	0	1	0	1	0
0x03	0	1	1	0	1	0	1
0x0B	0	1	1	0	1	0	0
0x19	1	0	0	1	1	0	0
0x1B	0	1	1	0	0	1	0

3 Universal programmable calculator (5 marks)

See the universal programmable calculator discussed in class.



a) Provide binary ROM contents that makes the universal programmable calculator calculate $4 \cdot (a + b)$ according to the comments. Provide X for bits that can be either 0 or 1. **(5 marks)**

C	D	M4	M3	M2	M1	Comment
XXXX X XX XXX	000	X X X	X X X	X X X	00	R0 = a
XXXXX XXX XX	001	X X X	X X X	X X X	01	R1= b
XXXXXXXXXX	000	000/001	001/000	001	11	R0 = a + b
0 000000100	00 1	XXX	XXX	000	11	R1 = 4
XXXXX XXXXX	000	000/001	001/000	011	11	R0 = 4 (a + b)

4 Assembly – Instruction sets (8 marks)

a) Provide assembly instructions for the universal programmable calculator we have discussed in class to calculate $a^2 + 3 \cdot c$. Note that a and c are inputs to the circuit. The result should be stored in register `R0`. **(6 marks)**

INPUT	R0, 0	R0 ← a
MUL	R0, R0, R0	R0 ← a^2
set	R1, 3	R1 ← 3
INPUT	R2, 2	R2 ← c
MUL	R1, R2, R1	R1 ← $3 \cdot c$
ADD	R0, R0, R1	R0 ← $a^2 + 3 \cdot c$

b) Do processors with load-store-architecture support instructions that add two variables in memory within a single cycle? Why? **(2 marks)**

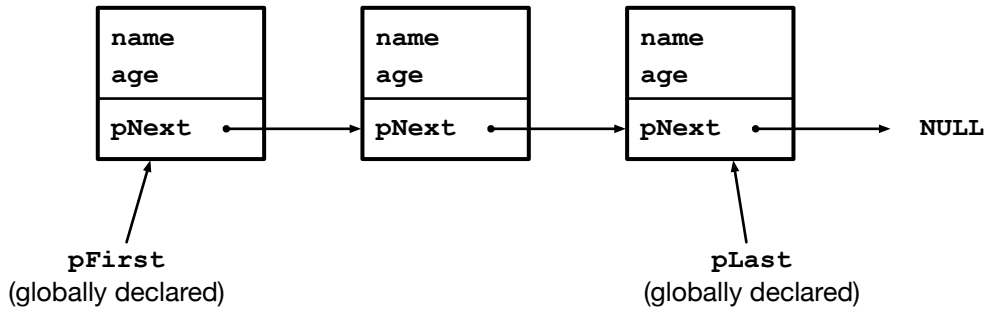
No, in a load-store-architecture, both operands to arithmetic instructions have to be located in registers

5 Singly linked list (7 marks)

See the following structure AgeList.

```
struct AgeList
{
    char name[100];
    int age;
    struct AgeList *pNext;
};
```

Two globally declared pointers pFirst and pLast of Type ''Pointer to AgeList'' provide entry points to a singly linked list as follows.



A new AgeList element has been created. `pEntry` of type ''Pointer to AgeList'' points to this new element.

a) Provide C code to add `pEntry` to the list in case the list is empty. **(3 marks)**

```
pFirst = pLast = NULL;
pEntry -> pNext = NULL;
```

b) Provide C code to add `pEntry` to the beginning of the list in case the list is not empty. **(2 marks)**

```
pEntry -> pNext = pFirst ;
pFirst = pEntry ;
```

c) Provide C code to add element `pEntry` between two elements, after an element `pPrevious` of type ''Pointer to AgeList''. Assume there is an element available following `pPrevious`. **(2 marks)**

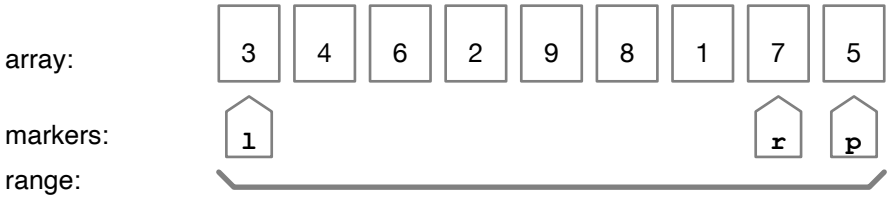
```
pEntry -> pNext = pPrevious -> pNext
pPrevious -> pNext = pEntry
```

6 The Quicksort algorithm (9 marks)

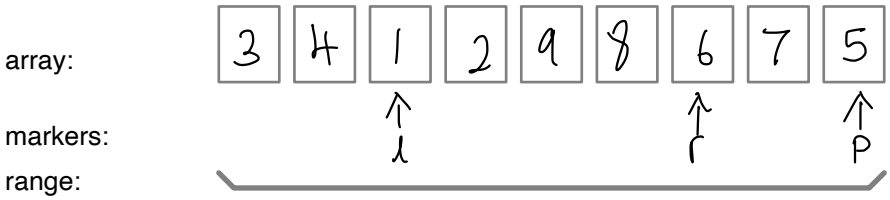
See the code of the quicksort algorithm we have discussed in class.

```
01 void quicksort(int a[], int left, int right)
02 {
03    int p, l, r, buf;
04
05    p = right;
06    l = left;
07    r = right - 1;
08
09    do
10    {
11        while(a[l] < a[p]) l++;
12        while(a[r] >= a[p] && l < r) r--;
13
14        if(l == r)
15        {
16            buf = a[r];
17            a[r] = a[p];
18            a[p] = buf;
19            p = r;
20        }
21        else if(l < r)
22        {
23            buf = a[r];
24            a[r] = a[l];
25            a[l] = buf;
26        }
27    } while(l < r);
28
29    if(left < p-1)
30        quicksort(a, left, p - 1);
31
32    if(p + 1 < right)
33        quicksort(a, p + 1, right);
34 }
```

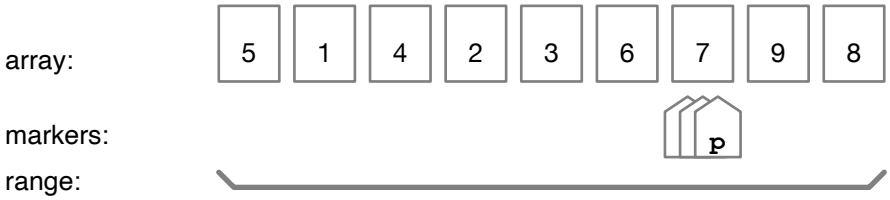
When function `quicksort` reaches line 9 for the first time, array `a` and markers `l`, `r`, and `p` have values as follows:



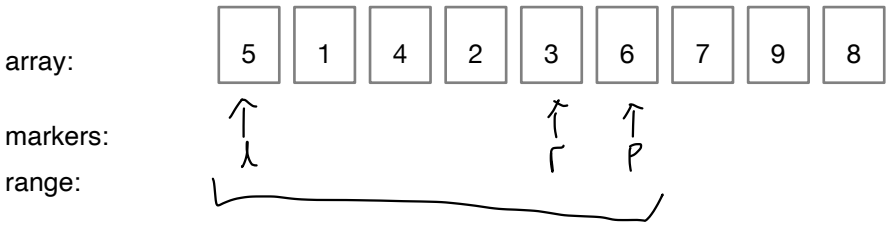
a) Provide values of array `a` and markers `l`, `r`, and `p` for the moment when execution reaches line 27 for the first time. **(5 marks)**



Given a different sorting of the array. When reaching line 30 for the first time, all markers point to element with value 7.



b) Provide the new range and markers `l`, `r`, and `p` for the moment when – in the second call of function quicksort – line 7 has just been executed. **(4 marks)**



7 Linear Search (9 marks)

See the following list of elements: 3, 6, 4, 8, 10, 9, 3, 2, 1, 7.

a) Provide the values of the first three elements that are analyzed when searching for value 2. **(3 marks)**

3, 6, 4

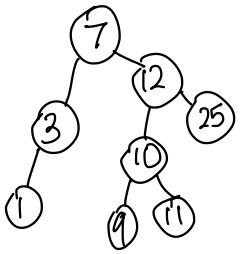
C function `int linear_search(int a[], int length, int number)` searches the number `number` in the array `a` of length `length` according to the linear search algorithm. If it finds the number, it returns the corresponding index. Otherwise, it returns -1.

b) Provide the missing C code to implement function `linear_search`. **(6 marks)**

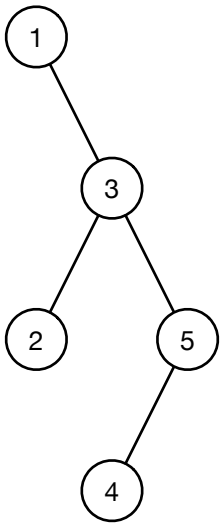
```
int linear_search(int a[], int length, int number)
{
    int i;

    for( i=0 ; i < length ; i++ )
    {
       if ( a[i] == number) return i ;


    }

    return -1;
}
```

8 Binary Search Tree (7 marks)

a) Provide the binary search tree for sequence 7, 12, 10, 11, 25, 3, 1, 9. **(4 marks)**



See the following binary search tree.



b) Provide the height of the tree. **(1 mark)**

3

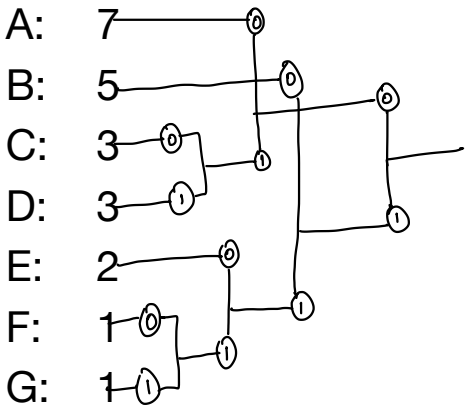
c) Provide two sequences that would result in this binary search tree. **(2 marks)**

1, 3, 2, 5, 4 1, 3, 5, 4, 2

9 Huffman Coding (9 marks)

a) Provide the Huffman tree for the following letter frequencies and the resulting Huffman code patterns in binary. Arrange 0 and 1 such that 0 is on top and 1 at the bottom of each branch. **(7 marks)**

Huffman tree:



Huffman code:

A: 00
B: 10
C: 010
D: 011
E: 110
F: 1110
G: 1111

Given the following Huffman encoding: A = 0, B = 10, C = 110, D = 111.

b) Provide the amount of bits it takes to code word ABAABCABD. **(2 marks)**

4x1 + 3x2 + 1x3 + 1x3

= 16 bits

Notes

Use this page for notes. If content on this pages sould be marked, indicate here and on the corresponding question space.

Notes

Use this page for notes. If content on this pages sould be marked, indicate here and on the corresponding question space.