EE23DP001
Aditya Shirodkar

- EE690 -
Embedded Systems Design
MidSem Paper
Assignment

Q1) a) Yes, as the credit card chip contains an ~~emd~~ embedded computer system for providing additional security during transactions.

Q1) b) Operating System Based: Digital Heads-Up Displays in cars
Infotainment systems in vehicles

Non-Operating System based: Vehicle ~~to~~ braking system
ECU for monitoring engine parameters

Q1) c) Yes. If the required functionalities can be achieved using analog ICs, then microcontrollers / microprocessors can be avoided.
eg) Analog Oscilloscopes, TL494 based PWM generation, OpAmp based comparators.

Q2a) Von Neumann

Q2b) 32 bit refers to the data bus width / ALU width

Q2c) a) 4 bit → Intel 4004
b) 8 bit → ~~E~~ 8085
c) 16 bit → 8086, AVR based
d) 32 bit → STM32, ~~Ar~~ Arm Corten M4 based

Q3a) Flash, EEPROM, ~~EPROM~~ ROM, PROM, UV-PROM

Q3b) 32 bit wide data bus; 17 bit wide address bus.

Q4) a) Instruction Set Encoding:
Identical, as all 3 controllers have the same core, i.e; Arm Cortex M4F

b) Memory Map:
Non identical, as memory mapping will depend on memory type and ~~by~~ size of memory ~~pai~~ paired with CPU.

c) Systick Registers:
~~Ext~~ Identical, as Systick is a part of the CPU ~~itself~~ core itself

d) Numbers of GPIO pins
Non Identical, as ~~t~~ number of GPIO pins may vary ~~with~~ with chip design

e) clock speed
Non identical, as even though the max clock speed of ~~at~~ ARM cortex M4 may be 80MHz, it can be lowered by chip manufacturer to meet application requirements.

Q5) Benefits of thumb mode:
1) Lesser memory needed to store ~~the~~ thumb instructions (16bit) as compared to 32 bit instructions.
2) Ability to fetch higher ~~number~~ number of thumb instructions as ~~com~~ compared to 32 bit instructions for a given bus width.

Q6) Relative Jump:

→ Relative jumps are limited to a smaller address range
→ They are to be used for jumping to address locations close to current address location
→ They are faster to ~~exe~~ execute

· Absolute Jump:

→ Absolute jumps are not limited by any address range, and can jump to any ~~location~~ location specified by the program
→ They are to be used for ~~jum~~ jumping to address locations present at any location in the code, and not just nearby locations
→ They are slower to execute.
→ They are to be used if a jump to a ~~loc~~ particular location is needed from different locations in the code.

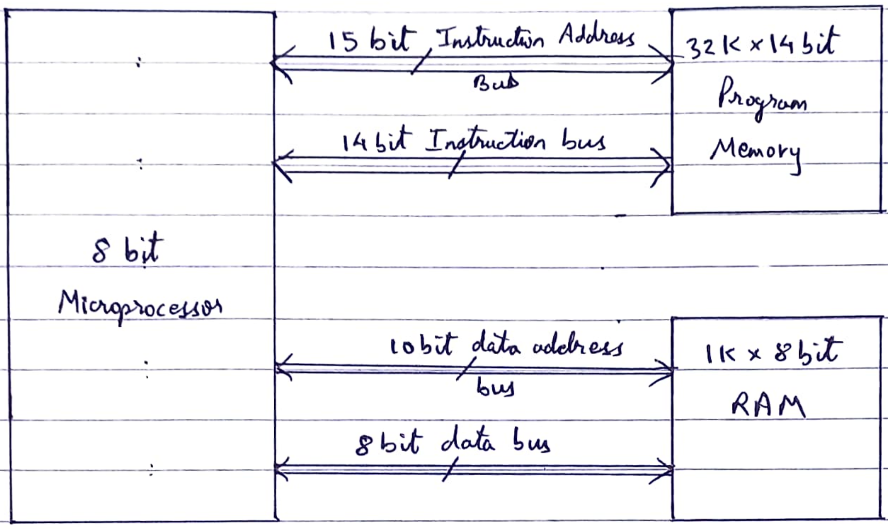
Q7) ~~0x3F →~~ ~~0x3D →~~

0x3F →	00
0x3E →	00
0x3D →	04
0x3C →	F8

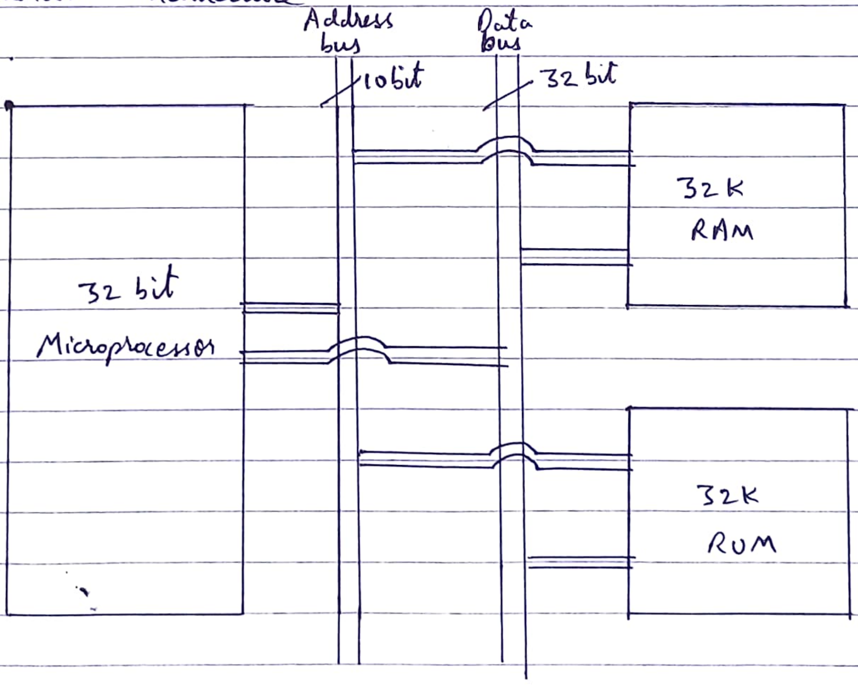
Q8) Embedded toolchain Components:
Editor, Compiler, Debugger, Assembler, Linker, Flasher

Q9) Hash number, Branch

Q10) Harvard Architecture



Von Neumann Architecture



32K = 10K × 32 bit   ~~32K = 10K × 32 bit~~

Q11) 16 bit ALU operating on 16 bit instructions with 16 registers (R_0 to R_{15})
Functions: Add, Subtract, And, Or, Not, XOR

Solution: Use hex instruction ~~coding~~, encoding, where the encoded instruction has the format 0x_ _ _ _
Let 0x_ _ _ _ be 0x abcd

a: Range = 0 - F, where 'a' is mapped to registers R_0 to R_{15} ~~and~~

eg: $a = 0 \rightarrow R_0$
$a = 1 \rightarrow R_1$
$a = 2 \rightarrow R_2$
$\vdots \quad \vdots$
$a = F \rightarrow R_{15}$

a holds operand 1

b: Range = 0 - F, where 'b' is mapped to registers R_0 to R_{15} similar to 'a'
b holds operand 2

c: Range = 0 - F, where 'c' is mapped to registers R_0 to R_{15} similar to 'c'
c holds destination register

d: Range = 0-5, where $d = 0 \rightarrow$ ADD
$d = 1 \rightarrow$ SUBTRACT
$d = 2 \rightarrow$ AND
$d = 3 \rightarrow$ OR
$d = 4 \rightarrow$ NOT
$d = 5 \rightarrow$ XOR

Note: for d = 4; NOT operation, the operation occurs only on operand 1 and operand 2 is ignored. The output is stored in destination register.

eg) Add R_0 and R_1; store in R_2
encoded instruction: ~~0x1020~~
0x0120

eg) XOR R_2, R_3; store in R_{14}
encoded instruction: 0x23E5

Bits:	[15:12]	[11:8]	[7:4]	[3:0]
	↓	↓	↓	↓
	Operand 1	Operand 2	Destination	Operation

Q12 Let Operand 1 be stored in ~~registers~~ locations W_3, W_2, W_1, W_0
Let Operand 2 be stored in locations X_3, X_2, X_1, X_0.
Let the sum be stored in locations Z_3, Z_2, Z_1, Z_0

Addition:
$$\begin{array}{rcccc} & W_3 & W_2 & W_1 & W_0 \\ + & X_3 & X_2 & X_1 & X_0 \\ \hline & Z_3 & Z_2 & Z_1 & Z_0 \end{array}$$

Let registers used for storing operands be $R_0 - R_{11}$
Let register R_{12} be used ~~to~~ store address of memory
Assume W_0, W_1, W_2, W_3, X_0, X_1, X_2, X_3, Z_0, Z_1, Z_2, Z_3 are stored in a continuous manner.

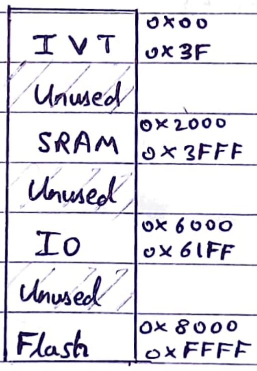
```
// R12 contains the address of register W0
~~Load R0 [~~
LOAD R0  [R12]   // Load W0
ADD  R12 #4      // Go to next memory location (W1)
LOAD R1  [R12]   // Load W1
ADD  R12 #4      // Next Location
LOAD R2  [R12]   // Load W2
ADD  R12 #4      // Next Location
LOAD R3  [R12]   // Load W3
ADD  R12 #4      // Next Location
LOAD R4  [R12]   // Load X0
~~ADD R5 [R12]~~
ADD  R12 #4      // Next location
LOAD R5  [R12]   // Load X1
ADD  R12 #4      // Next location
LOAD R6  [R12]   // Load X2
ADD  R12 #4      // Next location
LOAD R7  [R12]   // Load X3
```

R_{11}	Z_3
R_{10}	Z_2
R_9	Z_1
R_8	Z_0
R_7	X_3
R_6	X_2
R_5	X_1
R_4	X_0
R_3	W_3
R_2	W_2
R_1	W_1
R_0	W_0

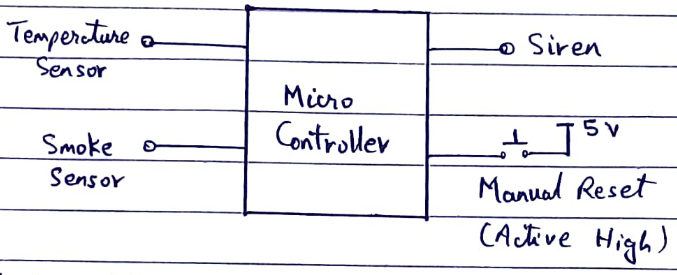
```
ADD   R12  #4       // Point to result
ADD   R8   R0  R4   // R8 = R0 + R4
ADDC  R9   R1  R5   // R9 = R1 + R5 + Carry
ADDC  R10  R2  R6   // R10 = R2 + R6 + Carry
ADDC  R11  R3  R7   // R11 = R3 + R7 + Carry

// store final result
STORE   R8    [R12]       // stores Z0
ADD     R12   #4
STORE   R9    [R12]       // stores Z1
ADD     R12   #4
STORE   R10   [R12]       // Stores Z2
ADD     R12   #4
STORE   R11   [R12]       // stores Z3
```

Q13) 16 bit memory ; 16 bit wide address bus
↳ max address = 2^{16} = 0xFFFF

Type	Start Address	End Address	Length
IVT	0 (0x0)	63 (0x3F)	64
Unused	64 (0x40)		
SRAM	0x2000	0x3FFF	8192
Unused	0x4000		
IO	0x6000	0x61FF	512
Unused	0x6200		
Flash	0x8000	0xFFFF	32k = 0x8000



Q14 Hardware Requirements: Temperature Sensor
Smoke Sensor
Manual Reset
Siren
Microcontroller

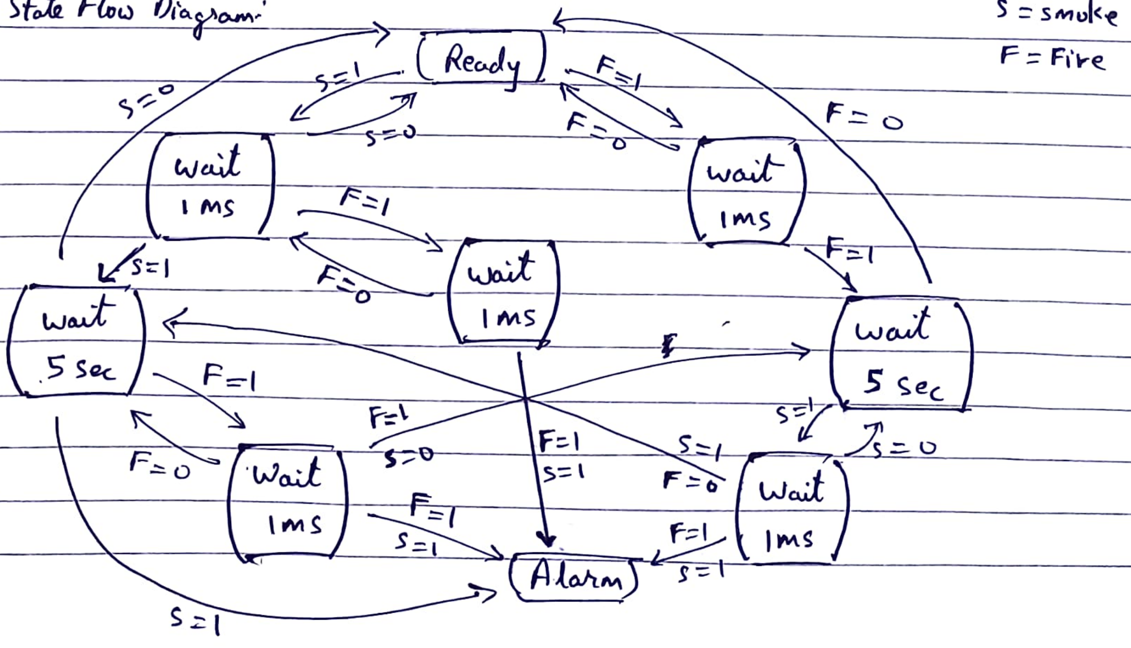


- High level description:
  - Smoke detected AND Fire detected → Siren immediately activates
  - Smoke detected OR Fire detected → wait for 5 seconds → Sensor still detects → Siren activated

Siren can be deactivated only through manual reset

On sensor activation, 1ms delay and recheck is done to prevent false positive based siren trigerring

State Flow Diagram:


S = smoke
F = Fire
Ready
S=1
S=0
F=1
F=0
Wait 1 ms
Wait 1 ms
Wait 1 ms
Wait 5 sec
Wait 5 sec
Wait 1 ms
Wait 1 ms
F=1
F=0
S=1
S=0
F=1
S=1
Alarm