2019 – 2020 Computer architecture paper

1. (a)(i)

E = (((A ∙ B)’ ∙ A)’ ∙ ((A ∙ B)’ ∙ B)’)’

(((A’ + B’) ∙ A)’ ∙ ((A’ + B’) ∙ B)’)’ De Morgans’

((A’ ∙ A + B’ ∙ A)’ ∙ (A’ ∙ B + B’ ∙ B)’)’ Distributive

((0 + B’ ∙ A)’ ∙ (A’ ∙ B + 0)’)’ Negation

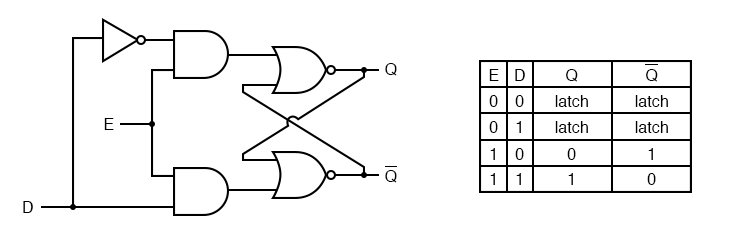
((B’ ∙ A)’ ∙ (A’ ∙ B)’)’ Simplification

(((B’ ∙ A) + (A’ ∙ B))’)’ De Morgan’s

B’ ∙ A + A’ ∙ B Negation

(ii) XOR gate

(b)

There is an invalid state for an SR latch when both inputs of the set and reset gate are 1. With a D latch, this eliminates the illegal state.

iii.

|  |  |  |
| --- | --- | --- |
| **Key** | **Synchronous Sequential Circuits** | **Asynchronous Sequential Circuits** |
| Definition | Synchronous sequential circuits are digital sequential circuits in which the feedback to the input for next output generation is governed by clock signals. | On other hand Asynchronous sequential circuits are digital sequential circuits in which the feedback to the input for next output generation is not governed by clock signals. |
| Memory Unit | Clocked flip flops are used as memory elements in synchronous sequential circuits. (Flip flops are circuits that are capable of storing one bit of information) | Time delayed devices are used as memory elements in asynchronous sequential circuits. They are implemented by a feedback in the circuit. In simple words, asynchronous sequential circuits are combinational circuits with feedback. |
| State | The states of Synchronous sequential circuits are always predictable and thus reliable. | On other hand there are chances for the Asynchronous circuits to enter into a wrong state because of the time difference between the arrivals of inputs. This is called as race condition. |
| Complexity | It is easy to design Synchronous sequential circuits | However on other hand the presence of feedback among logic gates causes instability issues making the design of Asynchronous sequential circuits difficult. |
| Performance | Due to the propagation delay of clock signal in reaching all elements of the circuit the Synchronous sequential circuits are slower in its operation speed | Since there is no clock signal delay, these are fast compared to the Synchronous Sequential Circuits |
| Example | Synchronous circuits are used in counters, shift registers, memory units. | On other hand Asynchronous circuits are used in low power and high speed operations such as simple microprocessors, digital signal processing units and in communication systems for email applications, internet access and networking. |

(c)(i)

1410 = 011102

310 = 000112

-310 = 111012

011102 + 111012

= ~~1~~010112 (Carry-out bit is discarded)

= 2310 + 2110 + 2010

= 1110

(ii) Taking the two’s complement representation of 14 and 3 from (i)

011102 + 000112

= 100012

There is an overflow as there are insufficient bits to represent 14 + 3 = 17.

(iii) –1410 = 100012 + 000012 = 100102

– 310 = 111002 + 000012 = 111012

100102 + 111012

= 1011112

There is an overflow because there are insufficient bits to represent –14 + –3 = –17.

(iv) From (i) and (iii), we have the two’s complement representation of 310 and –1410

–1410 – 310

= –1410 + –310’

From (iii) this will lead to an overflow.

(d)(i) 8G × 64bits = 8G × 8bytes = 64G

(ii) 8G / 512M = 16

64bits / 4bits = 16

16 × 16 = 256

(iii) Size of one RAM chip = 512M × 4bits = 256M

Number of chips in a memory module = 64bits/4bits = 16

Total number of bytes in a memory module = 256M × 16 = 28 × 220 × 24 = 232 bytes

32 bits are required.

(iv)

Method 1: convert to binary, extract relevant bits (see slide 46, lecture 4)

406610 = 1111111000102

Total number of address bits

= address bits within memory module + address bits between modules

= 32 + log216

= 36

Module bits are the first 4 bits in the address à 4066 is in module 0.

Method 2: divide by the number of addresses within a memory module – account for whether word/byte addressable (tutorial 2 question 4 vs. question 3)

4066 / 512 / 220 = Module 0

(v) Method 1:

From 1, 406610 = 1111111000102t

­Module bits are the last 4 bits à 4066 is in module 2.

Method 2:

4066 mod 16 = 2

Module 2

(e)(i)

41BA 3333 = 0100 0001 1011 1010 0011 0011 0011 0011

Sign: 0

Exponent: 1000 0011

Significand: 0111 0100 0110 0110 0110 011

4120 6666 = 0100 0001 0010 0000 0110 0110 0110 0110

Sign: 0

Exponent: 1000 0010

Significand: 0100 0000 1100 1100 1100 110

(ii) 1000 0010 is the smaller exponent, therefore its significand will be adjusted by 1

place: 1010 0000 0110 0110 0110 011

01.1010 0000 0110 0110 0110 011

+ 00.0111 0100 0110 0110 0110 011

10.0001 0100 1100 1100 1100 110

Sign: 0

The exponent has to be adjusted by 1 à 1000 0100

New significand: 0000 1010 0110 0110 0110 011

(iii) Exponent: 1000 0100 à 27 + 22 = 128 + 4 = 132

Since the exponent is excess-127, exponent = 132 – 127 = 5

1. (a)(i)

09H: 1F

0AH: 7A FE 06 36

(ii)

Assuming byte-addressable memory: 1K = 210 à 10 bits to represent

14 different operations à 4 bits to represent

Four general purpose register à 2 bits to represent

Instruction format 1 (16 bits):

Register-to-memory:

OP Reg1 Memory

4-bits 2-bits 10-bits

Instruction format 2 (16 bits):

Register-to-register:

OP Reg1 Reg2 Unused

4-bits 2-bits 2-bits 8-bits

(iii)

PUSH data onto the Top of Stack  
POP data from the Top of Stack

Stacks follow the  
Last-In, First-Out (LIFO) Rule:  
Last Data Pushed = First Data Popped

(iv)

EIP & EFLAGS on the stack. Once the interrupt completes, iret is called which restores the state by jumping to the return address on the stack (EIP).

(b) YYDS!

Sum = 0

Addr = something

Sum = Memory[addr]

Xpower = x

Loop exit when N <= 0

N = N – 1

Addr = Addr + 1

term = xpower

term = term × Memory[addr]

term = term + sum

sum = term

xpower = xpower × x

End loop

Register allocation:

Register 0 for x and xpower

Register 1 for addr

Register 2 for N

Register 3 for term and sum

Addr Assembler Instr Comment

1. 0 Zero
2. 1 One
3. 1F0H Start address
4. N number of items in array
5. X
6. Sum
7. xpower

…

10H LOAD R1, [2H] Addr

11H STORE R1, [5H] Sum = Memory[addr]

12H LOAD R2, [3H] N

13H **LOAD R0, [4H**] x

14H STORE R0, [6H] Initially, xpower = x, but subsequently, xn

15H SUB R2, [1H] N = N – 1

16H IFZER R2, 26H loop exit when N <= 0

17H IFNEG R2, 26H

18H ADD R1, [1H] Addr = addr + 1

19H LOAD R3, [6H] term = xpower

20H MULT R3, [R1] term = term × Memory[addr]

21H ADD R3, [5H] term = term + sum

22H STORE R3, [5H] sum = term

23H MULT R0, [6H] xpower = xpower × x

24H LOAD R0, [4H]

25H GOTO 14H

26H STOP

few questions

* + You add one to the address before your loop but then loop N times, doesn’t that put you out of range by 1 for the last iteration?
    - Good point.
  + Line 21H does R0\*R0 but this would actually mean R0 goes 1,x,x^2,x^4..., I feel like it maybe should be MULT R0, [4H]
    - Good point, edited.
  + Line 12H is storing the value in R1 to 5H but R1 is the start address 1F0H, so that would make the first term in the sum an address not the first value a0
    - I’m basing this off the examples in the slides and tutorial, but I’m assuming what happens is that instead of storing the address, it goes to the address and stores the value at that address.