2020-2021

Diagram

Description automatically generated

B (extension): MUX uses select lines to allow only one output, whilst decoder enable inputs to control the operation of the decoder.

C..draw out the table and then state:

D0 =A’.B’C’.D’

D4=A’.B.C’.D’

Diagram

Description automatically generatedFor iv, convert –ve to +ve for 2 complements. But the question says assume they are in two’s already?.

* That’s correct. It is already in two’s complement, the representation of +3 is the same in twos complement as it is in normal binary. So what is happening here is effectively saying: take +3 in binary (which is the same as how it is represented in twos complement), and make it negative (invert and add 1) to get the value –3 in twos complement. Sorry if this is poortly explained.

Diagram

Description automatically generated

Believe ive misunderstood iv) I think it is 31 bits

512M rows, 32bit/4byte per row => 2G =2^31 addresses needed

Agree ! Also Agree 😊

I think it should be module 15 not 16 for the high-order interleave (modules numbered from 0)

**Low and high order interleave qs**

**iv)** 2^35 = 100000000000000000000000000000000000

Module 16

Can also do 2^35 div 2^31 (module length of 512M \* 2^2) = 2^4 = module 16

**v)** 100000000000000000000000000000000000

Module 0

Can also do 2^35 mod 16 (no. modules) = module 0

A picture containing table

Description automatically generated

Minor point but looks like there’s a final number missing off the end of that last part, I got a 0 there for a final hex number of 428D4D66 - I got 428E4E66? +1 I also got 428e4e66, this is also result if you check online with a calc

I got the same answer 428E4E66

Part B

2.

a) i. 23 0b, F0 EF 74 CB

ii. register, immediate, memory

Both immediate and register should be faster than memory since the registers do not require access to main memory. Not sure how immediate and registers compare

Ideally immediate should be faster than register as the CPU would not need to travel to the register to grab the values, which is additional clock cycles and distance traveled.

However, this is usually not the limiting factor in CPU, and iso t depends.

iii. a) so CPU can transfer control to device’s interrupt handler

Used to service hardware timers, transfer data to and from storage (e.g. IO) and communication interfaces, handle keyboard and mouse events and to respond to any other time-sensitive events as required by the application system. (wiki)

Hardware Interrupt is caused by some hardware device such as request to start an I/O, a hardware failure or something similar. Hardware interrupts were introduced as a way to avoid wasting the processor’s valuable time in polling loops, waiting for external events.

For example, when an I/O operation is completed such as reading some data into the computer from a tape drive.

b) CPU checks if interrupt flag is set in each cycle

flaggy bois (EFLAG);

pushes contents of EIP reg onto stack

interrupt controllerw sends interrupt number to CPU

interrupt handler

The CPU first completes currents instruction when received interrupt from hardware via buses, or from CPU/software all via interrupt vector number. The interrupt service routine notifies the CPU of this value, which examines an interrupt descriptor table to map the lalue to an event handle, including the handler’s start address. The CPU then wakes the interrupt service thread to service the event.

c) context switch (see OS!) saves some stuff in a register I think

Push the EFLAGS Register and Return Address (i.e. contents of the EIP register) onto the Stack

Save program counter and registers on stack

d) re enable interrupts by setting interrupt enable flag in EFLAGS with sti

* **Disable interrupt** − Processor will ignore further interrupts while processing one interrupt. Interrupts remain pending and are checked after the first interrupt has been handled. In this process interrupts are handled in sequence
* **Define priorities** − In this method low priority interrupts can be interrupted by higher priority interrupts. Here, high priority interrupt will be handled then processor returns to previous interrupt on which it was earlier working.

b)

Pseudocode

Max = 0

Addr\_max = 0

I=N

While(I>0)

I--

If(A[I]>max)

max = A[I]

addr\_max = &A[I]

Memory:

000H 0

001H 1

002H 0 (max)

003H 0 (addr\_max)

004H A00H

005H N

A00H Start of A

Use of registers:

R0 i

R1 addr of current A[I]

R2 addr of max

R3 comp register

Start:

LOAD R0, [005H] I = N

LOAD R1, [004H] A[0]

ADD R1, [005H] A[N]

LOAD R2, [00H] addr\_max = 0 (I.e. max=0)

Loop:

IFZER RO, end if I =0 goto end

IFNEG RO, end if I <0 goto end

SUB RO, [001H] I=I-1

SUB R1, [001H] A[i-1]

LOAD R3, [00H] comp =0

ADD R3, [R1] comp = A[i]

SUB R3, [R2] comp = A[i] - max

IFNEG R3, loop if comp <0 (I.e. max > A[I]) goto loop

STORE R1, [003H]

LOAD R2, [003H] R2=R1, addr\_max= A[I] (as far as I can see there is no way to copy registers without accessing memory)

GOTO Loop

End:

LOAD R3, [R2] R3 = max

STORE R3, [002H] Store max at 002H

STOP