07.05.2020

Istanbul Technical University
Department of Computer Engineering

BLG 322E – Computer Architecture
Recitation 2

Note: If you have a question about the recitation, you may contact the research assistants of the course (esengun@itu.edu.tr).

QUESTION 1: DMA

For the four states in the instruction cycle of a CPU, their memory access requirements (if any) are given followed by their durations :

1. Memory access: 50 ns

2. No memory access: 40 ns

3. Memory access: 50 ns

4. Memory access: 50 ns

The memory access time and I/O interface access time are both 50 ns.

In this system, there are two 3-wire (BR, BG, BGACK) DMACs ($DMAC_1$ and $DMAC_2$). Both DMACs are of the **fly-by** (implicit) type (i.e., data does not pass through the DMAC).

The order of precedence is $DMAC_1 > DMAC_2$.

Assume that we start a clock (Clock = 0) when the CPU begins to run the program.

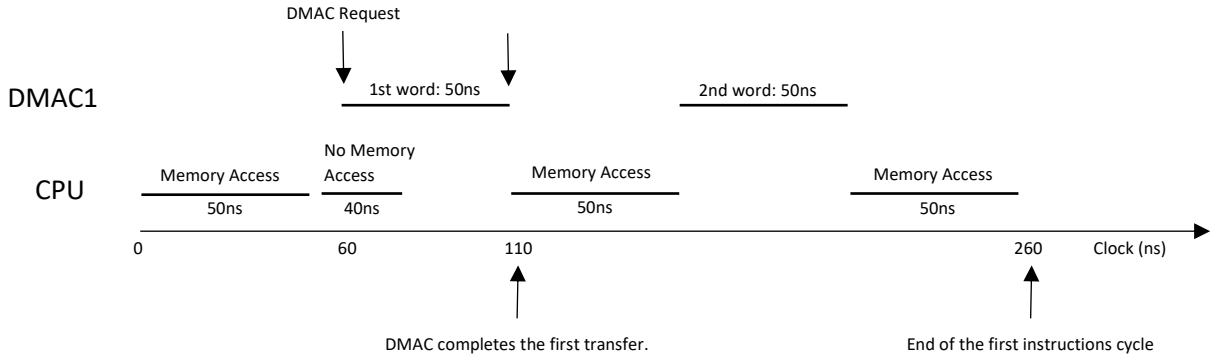
a) $DMAC_1$ attempts to start the data transfer for 5 words at Clock = 60ns using the **cycle-stealing** technique. Assume that the I/O interface is always ready to transfer. If there is no request from $DMAC_2$, when (Clock =?) will $DMAC_1$ complete the transfer of the first word? When (Clock =?) will the CPU complete the first instruction cycle?

b) If both controllers operate in **cycle-stealing** mode and both attempt to start the data transfer for 5 words at Clock = 60ns, when (Clock =?) will $DMAC_2$ complete the transfer of the first word? When (Clock =?) will the CPU complete the first instruction cycle?

c) Assume that $DMAC_1$ operates in **cycle-stealing** mode and $DMAC_2$ operates in **burst** mode. Both attempt to start the data transfer for 5 words at Clock = 60ns. When (Clock =?) will $DMAC_1$ complete the transfer of the second word? When (Clock =?) will the CPU complete the first instruction cycle?

SOLUTION 1: DMA

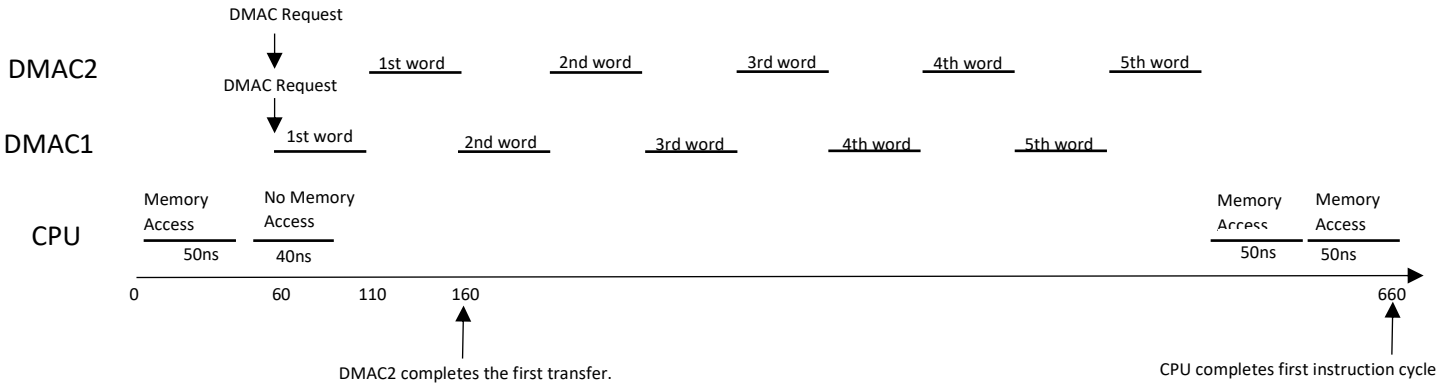
a)



$DMAC_1$ completes the first transfer at Clock = 50 + 10 + 50 = <u>110ns</u>.

The first instruction cycle will be completed at Clock = <u>260ns</u>.

b)



$DMAC_1$ has higher priority. $DMAC_1$ starts the first transfer at Clock = 60ns.

Because of the cycle-stealing mode, $DMAC_1$ releases the bus after the first transfer. Clock = 110ns

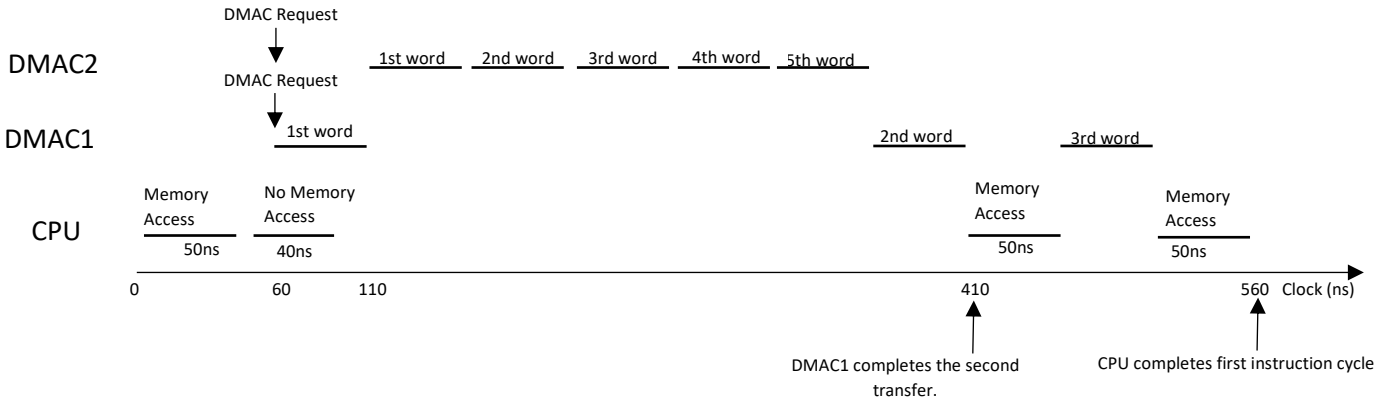
$DMAC_2$ gets the bus (DMACs have higher priority than the CPU)

$DMAC_2$ will complete the transfer of the first word at Clock = 110 + 50 = <u>160ns</u>

The CPU will complete the first instruction after all DMA transfers have been completed.

60 + (50 + 50) x 5 + 50 + 50 = <u>660ns</u>

c)



$DMAC_1$ operates in cycle-stealing mode, DMAC2 operates in burst mode.

$DMAC_1$ will transfer one word and release the bus (cycle stealing). Then, $DMAC_2$ transfers all words until the word counter is zero. $DMAC_1$ will get the bus and transfer the second word (Clock = 410ns).

Clock = 60 + 50 + 5x50 + 50 = <u>410 ns</u>

At Clock = 410 ns, the CPU gets the bus and accesses the memory for 50 ns.

At Clock = 460 ns, $DMAC_1$ gets the bus and accesses the memory for 50 ns.

At Clock = 510 ns, the CPU gets the bus and accesses the memory for 50 ns.

At Clock = <u>560 ns</u>, the CPU completes the first instruction.

QUESTION 2: TAS instruction

In a multiprocessor or multiprocess computer system, the code below is written to protect a critical section (shared memory).

```
CHECK     CMP 0, FLAG          Compare FLAG to zero
          BNZ CHECK            Branch if not zero to CHECK
          MOV $80, FLAG        FLAG=80 Hex
CRITICAL  …                    Start of the critical section
          …                    Access the shared memory
          CLR FLAG             Clear the FLAG
```

a) Why can this code not protect the critical section? Explain the situations causing the problem step-by-step in a system, where
   i. there are two parallel processes (P1 and P2).
   ii. there is a CPU and a DMAC.

b) Write proper code to protect the critical section using the "Test-And-Set" (TAS) instruction.

c) Explain how the TAS instruction protects the critical section from concurrent (parallel) access.

SOLUTION 2: TAS instruction

a)

i.

In the given code, FLAG being zero means that synchronization is available, and FLAG being ($80) means that synchronization is not available.

- Let us assume that initially the value of FLAG is zero, meaning that the synchronization variable is available.
- P1 reads the value of FLAG from memory when executing the CMP instruction and observes that the value of FLAG is zero.
- After P1 reads the value of FLAG but before it writes the new value of FLAG by running the MOV instruction to complete the synchronization operation, either P1 context switches out and P2 context switches in if time-sharing on a uniprocessor, or P2 executes on another processor in the multiprocessor system.
- P2 executes the CMP instruction and observes the value of FLAG as zero.
- When both of processes continue executing since both observed that the value of FLAG is zero, both will set the FLAG value to $80 and enter the critical section, thus causing a race condition.

The problem here is that the read-modify-write of the synchronization variable (FLAG) is not atomic because the two instructions CMP and MOV can be divided by system interrupts (process switch). This suggests that we need an atomic instruction to implement a correct synchronization mechanism.

ii.

In this case, since we are talking about synchronization, a process (P1) is running on the CPU to execute this code. Let us imagine that another process (P2) is running on a device and accessing the FLAG variable using DMA accesses.

- P1 reads the value of FLAG from memory when executing the CMP instruction and observes that the value of FLAG is zero.
- After P1 reads the value of FLAG but before it writes the new value of FLAG to complete the synchronization operation, the DMAC gets the bus and transfers the value FLAG to the memory of P2. Remember that DMA accesses are possible during the execution of an instruction.
- P2 reads the value of FLAG by executing CMP using DMA and observes the value of FLAG as zero.
- When both processes continue executing since both observed that the value of FLAG is zero, both will set the FLAG value to $80 and enter the critical section, thus causing a race condition.

The problem here is that the read-modify-write of the synchronization variable (FLAG) is divisible with DMA accesses, and this suggests that during read-modify-write of the synchronization variable, the bus must be locked to avoid DMA accesses to the synchronization variable.

b)

```
TEST        TAS FLAG       Tests the semaphore and sets it if necessary
            BMI TEST
CRITICAL    …              Critical section
            …
END         CLR.B FLAG     Semaphore is cleared (unlock)
```

c)

The TAS instruction is an atomic instruction, executes as a single operation, and performs the read-modify-write of the synchronization variable atomically.

- This instruction cannot be divided by interrupts.
- During the execution of the TAS, the AS (address strobe) remains asserted to lock the bus and to prevent other devices from accessing memory.

QUESTION 3: Interrupt

A CPU with an 8-bit data bus has an Interrupt Request input (**IRQ**) and an Interrupt Acknowledgment output (**INTA**). Both signals are active at "1". If the vectored/autovectored input (**VA**) of the CPU is "0", the CPU works with vectored interrupts, so that it reads the interrupt vector number after the acknowledgment of the interrupt, when its Data Acknowledgment input (**DACK**) is "1". The CPU also supports autovectored interrupts. After the acknowledgment of an interrupt request (**INTA=1**), if the vectored/autovectored input (**VA**) of the CPU is set to "1", then the CPU does not read a vector number and works in autovectored mode.

In this system, there are four interrupt sources (*A1, A2, B1,* and *B2*) of two different types (*A* and *B*):

- Type **A** (A1 and A2): Each one of these devices has an Interrupt Request output (**IRQ**) and an Interrupt Acknowledgment input (**INTA**). They do not have vector number outputs. They work in autovectored mode.
- Type **B** (B1 and B2): Each one of these devices has an Interrupt Request output (**IRQ**), an Interrupt Acknowledgment input (**INTA**), and an 8-bit vector number output (**VN**).

Priority (precedence) order of the devices: A1 > A2 > B1 > B2

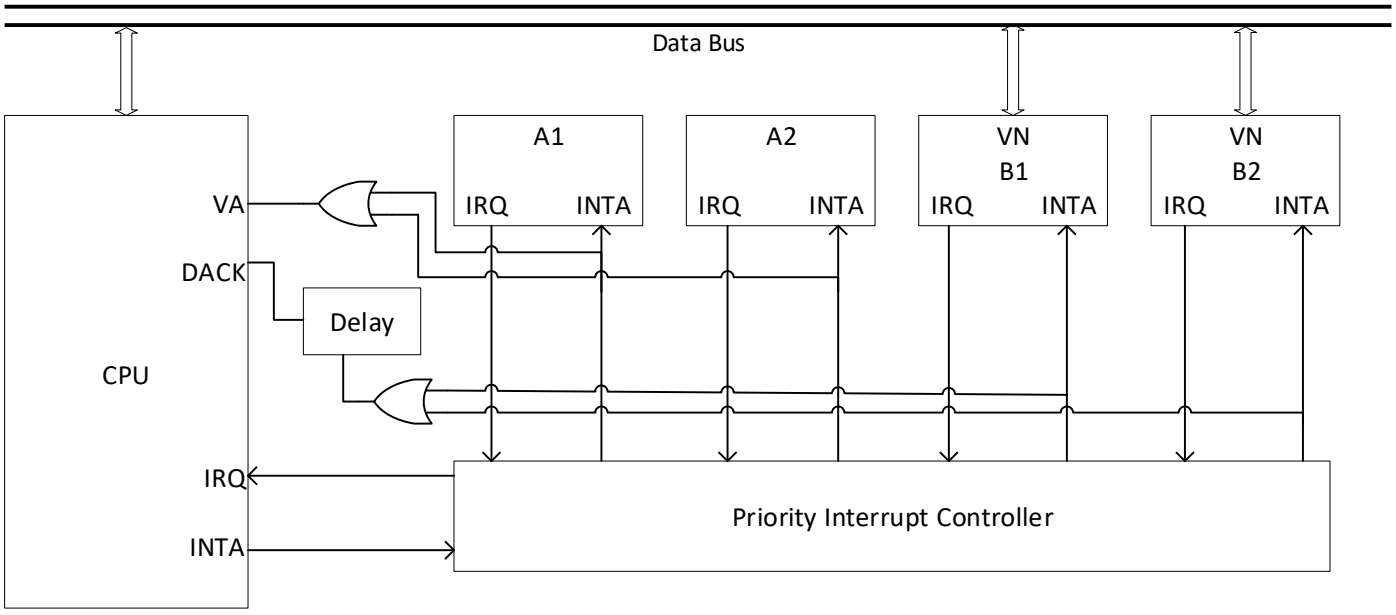
a) Design and draw the system with the CPU, the four devices (A1, A2, B1, and B2), and the priority interrupt controller. First, show the priority interrupt controller only as a black box. Then, design and draw the internal structure of the priority interrupt controller using logic gates.

b) Assume that devices *A1* and *B1* assert their interrupt requests at the same time. Show all the signals that are sent in the system step-by-step until requests of both devices have been fulfilled.

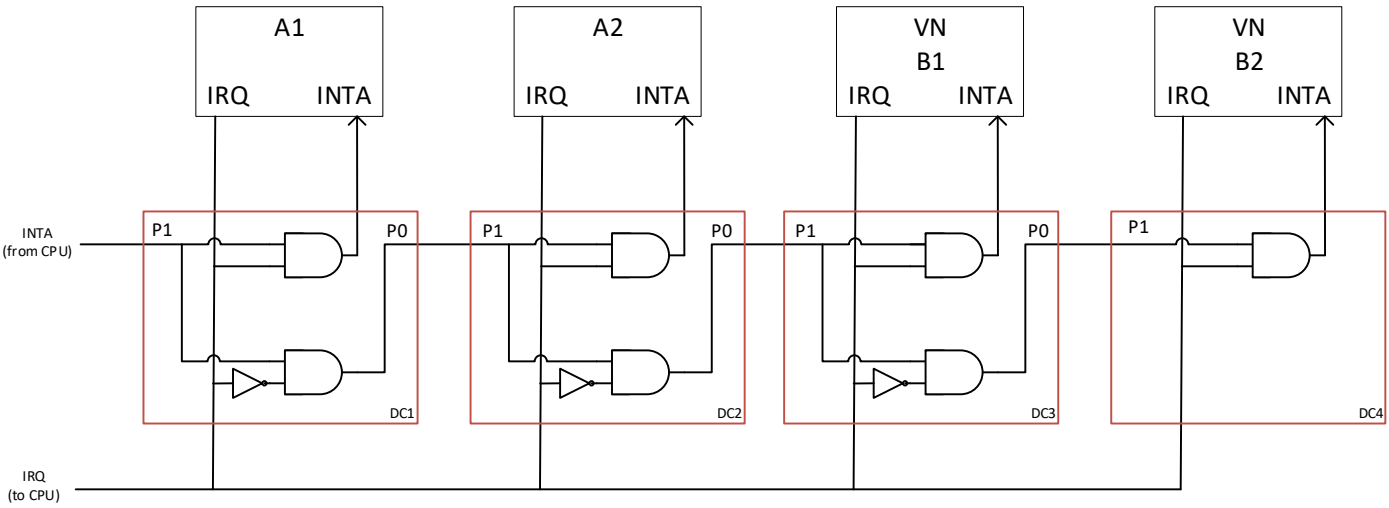
c) How does the CPU determine the start address of the interrupt service routine that will be run if the interrupt source is a device of type **A** or of type **B**?

SOLUTION 3: Interrupt

a)



Priority Interrupt Controller (Daisy Chain)



b)

IRQ(A1) = 1, IRQ(B1) = 1

IRQ(CPU) = 1

INTA(CPU) = 1 (Interrupt request accepted)

PI(DC1) = 1, PO(DC1) = 0, INTA(DC1) = 1 (Source of the request is A1)

INTA(A1) = 1, VA = 1. The CPU determines that the interrupt is autovectored from VA and checks the flags of A1 and A2 (software-based polling). Then, it determines that the source is A1. The request of A1 is removed (IRQ(A1) = 0). The ISR jumps to the procedure of A1 and returns after the operation.

IRQ(A1) = 0, IRQ(B1) = 1, IRQ(CPU) = 1

INTA(CPU) = 1 (Interrupt request accepted)

PI(DC1) = 1, PO(DC1) = 1 (Source of the request is not A1)

PI(DC2) = 1, PO(DC2) = 1 (Source of the request is not A2)

PI(DC3) = 1, PO(DC3) = 0 (Source of the request is B1)

INTA(B1) = 1, VA = 0 (Vectored interrupt. The vector number is put on the data bus)

IRQ(B1) = 0 (B1 removes request)

DACK = 1. CPU reads vector number of B1, gets the start address of the ISR from the vector table, and runs the ISR of B1. The ISR returns to the main program.

c)

Type A devices do not supply the CPU with a vector number, and interrupts are autovectored. In order to determine the source of the interrupt request, the CPU checks the flags of type A devices (software-based polling) in the interrupt service program for the autovectored interrupts. The CPU decides that the interrupt is autovectored based on the value of VA input. After determining the source of the request, the CPU jumps to the program of the source device.

Type B devices put the vector number on the data bus. After determining that the interrupt is vectored and acknowledging the interrupt, the CPU reads the vector number and fetches the beginning address of the corresponding ISR from the vector table. Then, the CPU jumps to this ISR.