Istanbul Technical University

Department of Computer Engineering

BLG 322E Computer Architecture
Homework 3 Report

Mustafa Can Çalışkan

Student ID: 150200097

May 4, 2025

Contents

1. Assumptions

Before starting the timeline construction and solution of the questions, the following assumptions were made in order to resolve ambiguities and ensure consistency with the underlying CPU and system behavior:

A1. No Housekeeping Required When Returning from Nested ISRs: In the case of nested interrupt service routines (ISRs), when control returns from an inner interrupt back to the outer one, no additional interrupt housekeeping operations are performed. This is based on the understanding that the initial housekeeping (such as saving the program counter and status register) was already completed when entering the outer ISR.

A2. TAS Treated as a Normal Instruction for Event Resolution: The `TAS` instruction is considered part of the main program and is indistinguishable from other instructions until it reaches the instruction fetch and decode stages. Therefore, any pending interrupt or DMA request is assumed to be handled *before* the actual execution of the `TAS` instruction begins. This assumption aligns with the system's standard behavior of checking interrupt lines after each instruction is completed.

2. Timeline Diagram

The timing and control diagram below has been constructed based on the initial conditions and events described in the assignment file. The timeline is derived from the system behaviors of interrupt processing, Direct Memory Access (DMA) operation, and instruction execution cycles of the CPU.

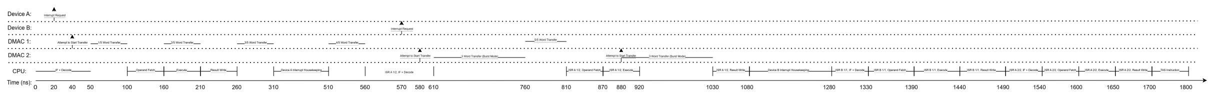

Figure 1: Timeline diagram of CPU, DMA, and interrupt events

This figure visually represents the sequence and overlap of various CPU, DMA, and interrupt activities. It is used as the main reference in answering the questions in the following sections.

3. Question 1

a) DMAC1 Transfer Completion Times

DMAC1 is configured to work in “cycle-stealing mode“, transferring one word at a time and allowing the CPU to continue execution between memory accesses. According to the specifications:

- Each DMAC1 memory access takes 50 ns (since memory access time is 50 ns). - DMAC1 cannot operate during CPU memory access periods (i.e., instruction fetch, operand fetch, result write, and interrupt housekeeping). - The first DMAC1 attempt occurs at “Clock = 40 ns“.

By analyzing the timeline and identifying idle memory slots, the approximate word completion times for DMAC1 are:

- **1st word:** 100 ns
- **2nd word:** 210 ns
- **3rd word:** 310 ns
- **4th word:** 560 ns
- **5th word:** 810 ns

These moments reflect the earliest opportunities where memory is available and not occupied by the CPU’s fetch/write/interrupt stages, and where DMAC1 can safely perform transfers.

All these are also marked in the timeline diagram (Figure 1).

b) DMAC2 First Attempt Completion Time

DMAC2 is configured in “burst mode“ and attempts its first transfer at “Clock = 580 ns“. Each word transfer takes 50 ns, and it transfers 3 words in one burst.

DMAC2 has “higher priority“ than DMAC1 and can occupy the memory bus exclusively during burst transfers.

- Total time for 3-word transfer: $3 \times 50\,\text{ns} = 150\,\text{ns}$
- DMAC2 must wait for any ongoing CPU memory usage to finish before it can take control of the memory bus.
- Based on the timeline, DMAC2 successfully starts after memory is freed around “Clock = 610 ns“.
- Thus, transfer ends at: $610 + 150 =$ `760 ns`

This time is marked on the timeline diagram.

c) DMAC2 Second Attempt Completion Time

The second attempt of DMAC2 occurs at “Clock = 880 ns“. Since DMAC2 uses “burst mode“, it requires uninterrupted access to memory for 3 consecutive words.

- Each word transfer takes 50 ns, totaling $3 \times 50 = 150$ ns.
- Based on the timeline, there is no conflict with CPU memory activity at this point.
- Therefore, DMAC2 can begin its transfer shortly after “Clock = 880 ns“ and finish at:

$$880 + 150 = \texttt{1030 ns}$$

This moment is marked clearly on the timeline diagram.

d) Completion Time of ISR for Device A (Triggered at “Clock = 20 ns“)

Device A sends an interrupt request at “Clock = 20 ns“. However, due to the CPU executing instructions and potential memory contention with DMAC1, the servicing of the interrupt is delayed.

Key details:

- “Interrupt Housekeeping“ requires 200 ns and cannot overlap with DMA transfers.
- The ISR for device A consists of 2 instructions.
- Each instruction has:
  - Instruction fetch/decode: 50 ns
  - Operand fetch: 60 ns
  - Execution: 50 ns
  - Result write: 50 ns
  - Total per instruction: 210 ns
- Total time: $200 + 2 \times 210 = 620$ ns

Taking into account when the CPU can begin servicing the interrupt and memory availability, the full ISR for device A is completed by “Clock = 1700 ns“.

This point is also highlighted in the timeline diagram.

e) Completion Time of ISR for Device B (Triggered at “Clock = 570 ns“)

Device B sends an interrupt request at “Clock = 570 ns“ and has “higher priority“ than device A. Since the CPU is executing the ISR of device A at that time, and nested interrupts are allowed, the CPU saves the context of device A and immediately begins the ISR for device B.

Details of the ISR for device B:

- “Interrupt housekeeping“: 200 ns
- 1 instruction:
  - Instruction fetch/decode: 50 ns
  - Operand fetch: 60 ns
  - Execution: 50 ns
  - Result write: 50 ns
  - Total per instruction: 210 ns
- Total: $200 + 210 = 410$ ns

Assuming device B’s ISR begins at approximately “Clock = 1080 ns“, it will complete at:

$$1080 + 410 = \texttt{1490 ns}$$

Once the ISR for device B is completed, the CPU returns to continue and finish the interrupted ISR of device A.

This interrupt nesting and preemption behavior is reflected in the timeline diagram.