Tom Robinson

Unit 19, Assignment 3

The Low-Level Programmer

Solihull College
November 2022

Task One P7

Polling & Interrupts

Polling

Polling is the process wherein a CPU or I/O Controller checks a device or array of devices on a regular basis to see whether it requires attention.

This process is less efficient when compared to interrupts since every device is checked at each polling cycle, even if it has no new information and no action needs to be taken.

USB is an example of a protocol that makes use of polling.

Interrupts

Interrupts to the CPU are issued by a device or program. Depending on their priority, they will delay the execution of the currently running program until they are completed.

Interrupts are issued on the control bus. When an interrupt is issued, the program counter of the previously executing program is stored in the stack register for the CPU to return to.

Both hardware and software can issue an interrupt. A hardware interrupt is issued when there is a memory parity error, a power button is pressed or other physical happenings. Software interrupts occur in software and are handled by an operating system's kernel. They can be triggered by a user authentication request that takes priority or a program performing an illegal operation.

I/O interrupts are issued when a buffer is running out of space, for example when sending data to a printer or hard disk drive. A DMA controller issues interrupts when a data operation—like a move or copy—is completed.

Task Two P8

RISC & CISC

Instruction Sets

An Instruction Set is the available commands within the CPU's machine language. A broader instruction set may facilitate easier low-level software development however it adds complexity to the CPU's design.

Reduced Instruction Set Computing

A RISC CPU is a processor that contains a limited set of instructions.

RISC processors present several advantages. They have a smaller die size and are more power efficient with a lower TDP (thermal design power). Additionally, they can run certain instructions in parallel in a single clock-cycle with pipelining. This efficiency makes them ideal for battery powered devices such as laptops, tablets, and smartphones.

The smaller set of instructions makes for longer machine language programs. A task that may be one instruction in a CISC chip may take several instructions on a RISC chip. This leads to more complicated programming.

A prevailing example of RISC computing is ARM-based processors, including the new Apple Silicon series. CPUs in the ARM family have been used since its creation in 1985 and early computers in the 1960s have been credited as the first RISC systems in general.

Complex Instruction Set Computing

CISC CPUs contain more instructions than their RISC counterparts, making them easier to program for at a lower level.

The additional instructions add complexity to the processor die and the manufacturing process, making CISC chips larger and more costly to produce. They are less efficient and often have significantly higher TDPs when compared to similarly performing ARM chips.

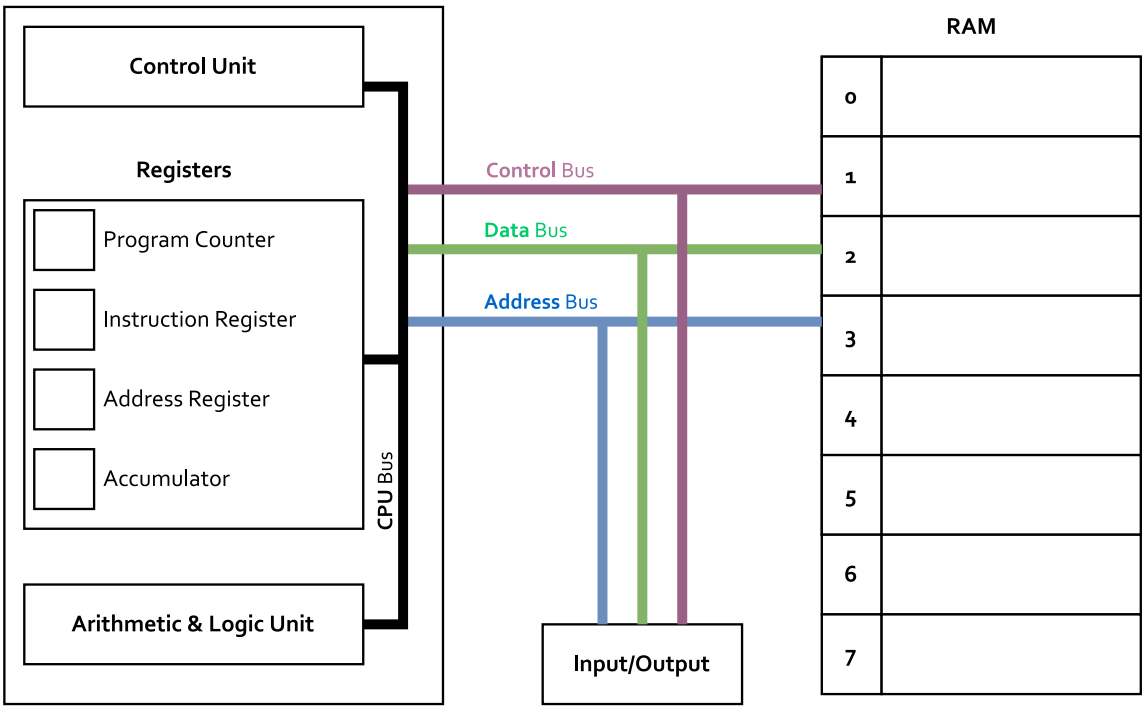
Modern examples of CISC CPUs include those using the X86 instruction set from AMD and Intel.

Task Three P9

FDE Cycle & Registers

The Fetch-Decode-Execute cycle describes the steps that are carried out by a CPU to run programs. Bolded text refers to registers within the CPU being used.

The Cycle



Fetch

In the fetch part of the cycle, the CPU fetches from the memory address stored in the **Program Counter** register.

The contents of this address are sent from memory—either cache or RAM/ROM—using the data bus. In the case of RAM or ROM, the CPU specifies the memory address using the address bus. The contents are stored in the **accumulator.**

Decode

The CPU translates the Opcode to its corresponding action. This could be reading from memory or performing arithmetic.

Execute

The CPU executes the instruction. The output of arithmetic operations is stored in the **accumulator.**

The **program counter** is increased or modified, and the CPU returns to the fetch stage of the cycle.

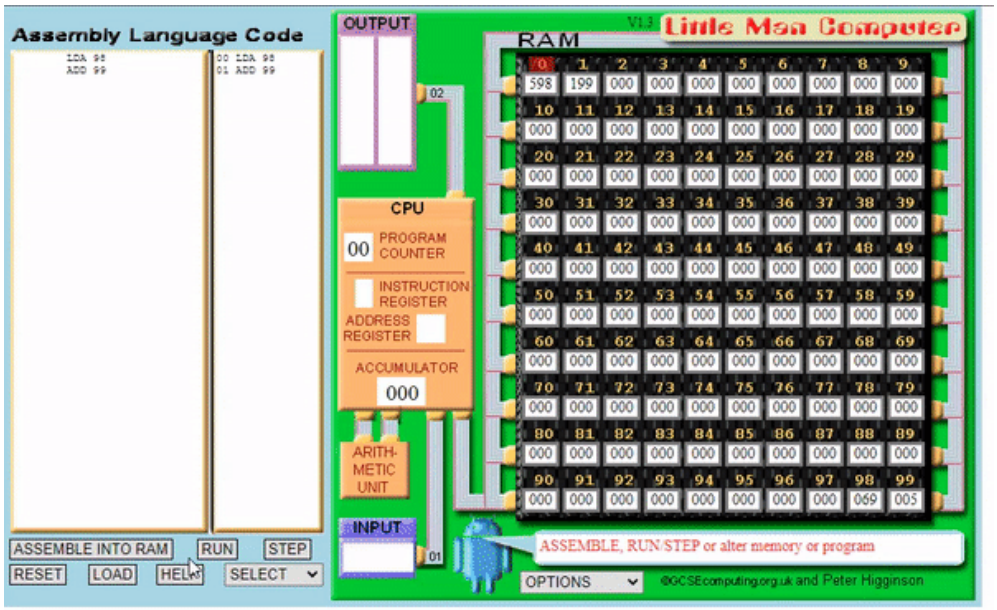
Polling & Interrupts

Peripherals connected to the CPU, be them directly connected or through an I/O controller, make use of polling and interrupts to communicate.

In the case of polling, each device has a specified polling rate. At this specified interval, the CPU checks the device to see whether any action must be taken.

Depending on their priority, interrupts delay the execution of the currently running program or may run afterwards. They are used to inform the CPU that an action is completed, or a significant event has occurred.

Adding Numbers



LMC-Addition.mp4

The first number is loaded into the **accumulator**. The ALU is then called by an ADD operation, which fetches the second number from memory and then adds it the first number in the **accumulator**. The output of the ALU's calculation is stored in the accumulator.

Task Four M3

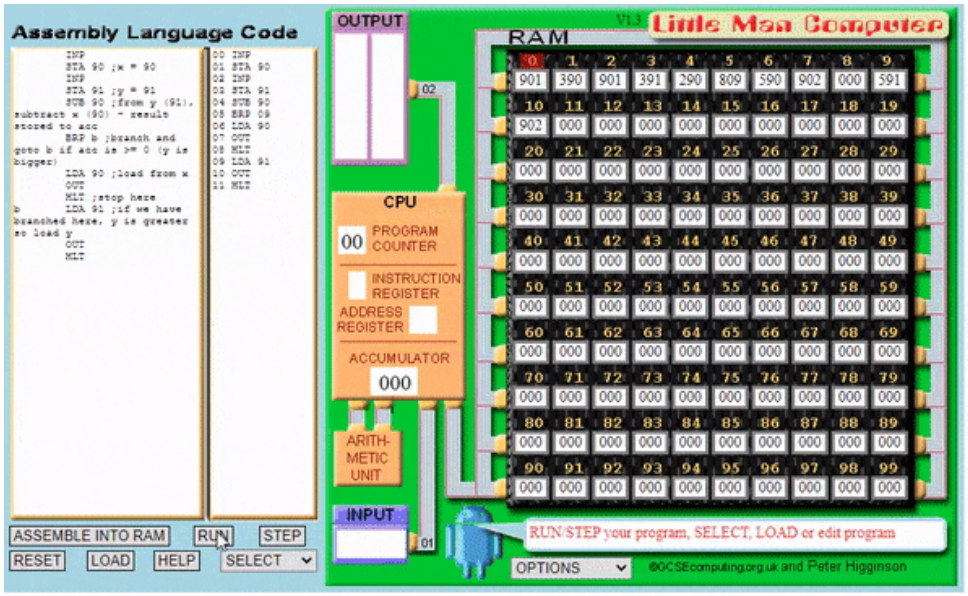
Assembly Program

```
1       INP
2       STA 90  ;x = 90
3       INP
4       STA 91  ;y = 91
5       SUB 90  ;from y (91), subtract x (90) - result stored to acc
6       BRP b   ;branch and goto b if acc is >= 0 (y is bigger)
7       LDA 90  ;load from x
8       OUT
9       HLT     ;stop here
10  b   LDA 91  ;if we have branched here, y is greater so load y
11      OUT
12      HLT
```

Branching.asm

The above assembly language program—written for the Little Man Computer simulator at peterhigginson.co.uk/LMC—compares two numbers and then outputs the larger one. The numbers are provided by user input and then a branch statement is used to decide which is shown based on the results of a subtraction.

The original source code file is attached to the right of the screenshot.



Branching Demo.mp4

The demonstration above shows the code running in the Little Man Computer. In this example, the two numbers inputted are 50 and 69. The program is successful in outputting the bigger of the two: 69.

The original screen recording is attached to the right of the embedded animated GIF.

Task Five D2

Bus Widths

A bus is made of a series of parallel wires – the number of wires dictates the number of bits that can be sent at once. The explanations below refer to the bus widths in the context of a CPU and the fetch-decode-execute cycle.

Address Bus

In the context of an address bus, increasing the width of the bus allows more memory to be accessed. 4-bit memory busses facilitate the addressing of only 16 (2^4) locations. A 32-bit memory bus allows addressing 4,294,967,296 (2^{32}) locations. Assuming each memory location contains a byte of information, this allows the addressing of 4.2 Gigabytes of data.

Data Bus

Each operation is broken up into three parts – the opcode, the addressing mode, and the operand. With an 8-bit opcode, a CPU can have 256 (2^8) instructions. With an addressing mode and the operand taking up 12 more bits, we would need a 20-bit data bus to send an instruction.

As with the address bus, the width of the data bus dictates how many bits can be sent to the CPU per cycle. When a modern CPU is called '64-bit,' this refers to the data bus width.

Being able to send more data per cycle increases the speed of a system.

Complexity

Each additional 'lane' of data in a bus needs to go somewhere. Wider buses that can transmit more data take up more physical space. It is for this reason that modern "64 bit" CPUs only actually make use of 48 or 52 bits. This is sufficient for accessing an address space of hundreds of terabytes. A full 64-bit bus would be superfluous for most applications and would add significant cost and manufacturing complexities to a CPU.