

Lab 6 - Building a Memory Hierarchy

In this lab you will be adding a memory sub-system for the single-cycle CPU you built previously. Some systems store both instructions and data in the same memory device, while other systems use separate memory devices for instructions and data. For our system, we will use separate memory devices (known as Harvard architecture). The lab will be completed in three parts.

Part 1 - Data Memory

[25 marks]

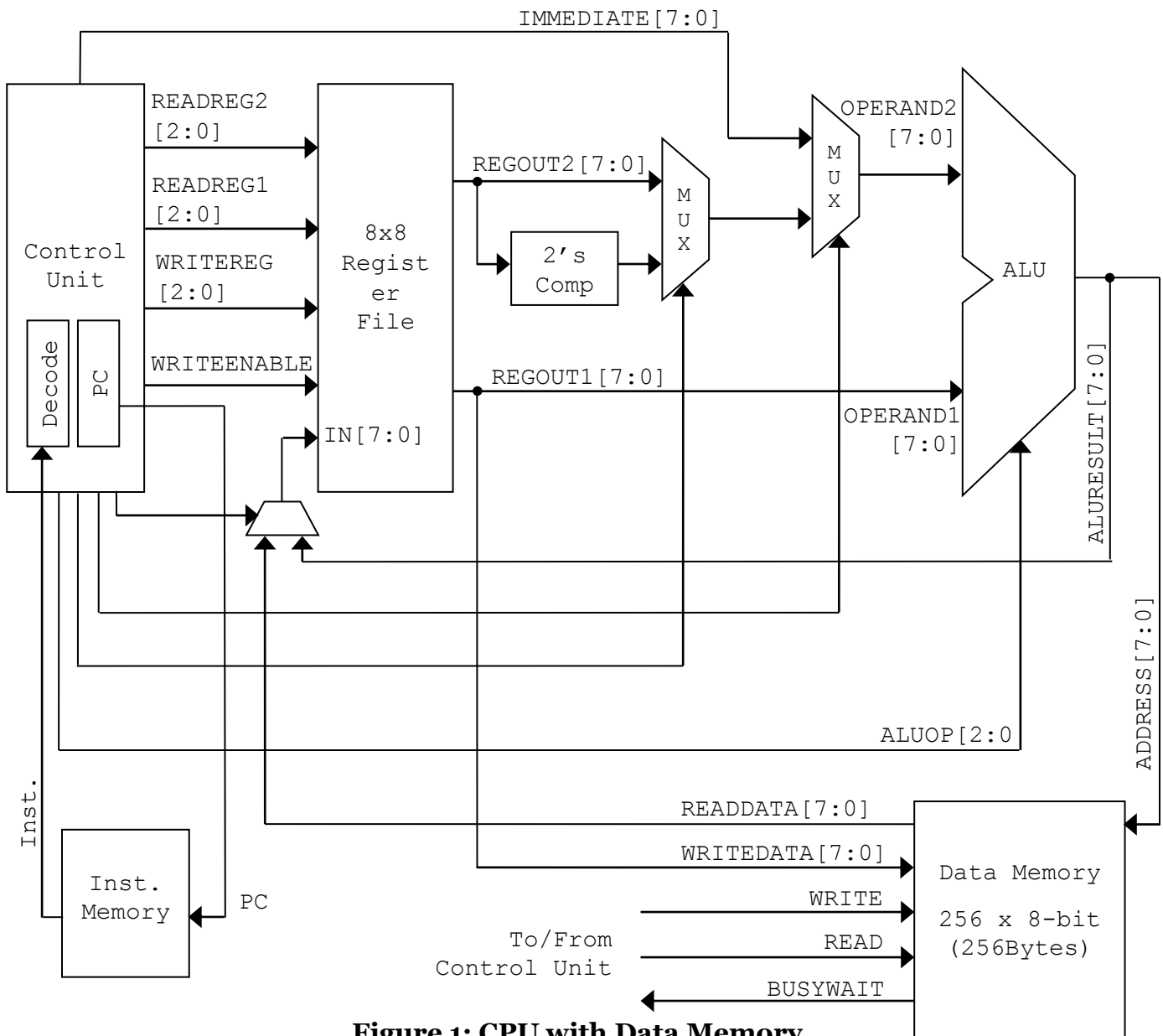


Figure 1: CPU with Data Memory

The diagram in Figure 1 shows a 256 Byte Data Memory module being connected to the CPU you built in the previous lab (note that the hardware for flow control instructions are not shown). A sample memory module is given to you which uses 256 8-bit registers to store data. The memory module uses the following signals to interface with the processor:

ADDRESS: Location in memory being accessed (read/write). ALU provides the value for this signal.

WRITEDATA: Data value to be stored in the memory, at the location pointed to by ADDRESS. This value is supplied by the Register File and received by the memory on a positive clock edge.

READDATA: Data value read from the memory, at the location pointed to by ADDRESS. This value is sent by the memory on a positive clock edge to the Register File for storing.

READ: Control signal to request the memory to perform a read operation on the provided ADDRESS. This signal is supplied by the CPU control unit.

WRITE: Control signal to request the memory to perform a write operation on the provided ADDRESS. This signal is supplied by the CPU control unit.

BUSYWAIT: Memory asserts this signal when CPU sets READ/WRITE control signals, and keeps it asserted while the operation is in progress. CPU control unit should stall the processor and hold the ADDRESS and READ/WRITE control signals stable while this wait signal is asserted. Memory de-asserts BUSYWAIT when a reading or writing operation is concluded. The next instruction should not be fetched by CPU until BUSYWAIT is de-asserted by the memory.

Study the given memory module and see how to connect it to your CPU. Note that a latency of 5 CPU clock cycles (#40 time units) is artificially added to the read and write operations inside the memory module in order to simulate it with realistic timing.

You need to implement hardware support for four new instructions (`lwd`, `lwi`, `swd` and `swi`) in your CPU, to use the data memory. The new instructions will follow a similar encoding format as previous instructions.

OP-CODE (bits 31-24)	RD (bits 23-16)	RT (bits 15-8)	RS/IMM (bits 7-0)
-------------------------	--------------------	-------------------	----------------------

In the new instructions, memory accessing can be done using two different addressing modes: register direct addressing; or immediate addressing. See examples below:

`lwd 4 2` : Read memory at address given in register 2 (RS) and store result in register 4 (RD). Ignore bits 15-8

`lwi 4 0x1F` : Read memory at address 0x1F (IMM) and store result it in register 4 (RD). Ignore bits 15-8

`swd 2 3` : write value from register 2 (RT) to the memory at address given in register 3 (RS). Ignore bits 23-16

`swi 2 0x8C` : write value from register 2 (RT) to the memory at address 0x8C (IMM). Ignore bits 23-16

Datapath timing for the new memory access instructions are given below. Note that Data Memory Access time is based on ideal caches. Your system doesn't have caches in part 1 and accessing memory directly will incur much higher delays, consequently stalling the CPU for several clock cycles.

lwd (register direct addressing):

PC Update	Instruction Memory Read		Register Read	ALU	Data Memory Access
#1	#2		#2	#1	#2
	PC+4 Adder		Decode		
	#1		#1		
Register Write					
#1					

lwi (immediate addressing):

PC Update	Instruction Memory Read			ALU	Data Memory Access	
#1	#2			#1	#2	
	PC+4 Adder		Decode			
	#1		#1			
Register Write						
#1						

swd (register direct addressing):

PC Update	Instruction Memory Read		Register Read	ALU
#1	#2		#2	#1
	PC+4 Adder		Decode	#2
	#1		#1	

swi (immediate addressing):

PC Update	Instruction Memory Read		Register Read	Data Memory Access
#1	#2		#2	#2
	PC+4 Adder		Decode	ALU
	#1		#1	#1

1. Connect the given *data_memory* module to your *cpu* as specified, via a testbench. Implement the new instructions, and modify the *cpu* accordingly. Make sure to stall the processor when the `BUSYWAIT` signal is asserted by the Memory.
2. Test your processor and memory with software programs containing the new instructions (in addition to the instructions from Lab 5). Program can be hardcoded inside the testbench, or loaded from a file. Store/load data values in the data memory via your program.
3. Submit a compressed file ***eXYYYY_lab6_part1.zip*** containing your Verilog files with all the modules in your design, testbench, and screenshots of timing diagrams clearly showing signals related to memory accesses.

To be continued...