EEE339/336 Solution Sheet 3

1. Because the address register is used not only for fetching program instructions but also for fetching data. The address of the datum has to be stored somewhere... and the AR register is a convenient place. In addition, if we fetched only program instructions from external memory, latching the address in AR allows the contents of the program counter (PC) to be incremented at the same time as the instruction fetch, thereby shortening the time taken for every instruction.

2. We need a buffer to *drive* the memory during a write operation because, invariably, the memory is on another chip from the CPU. We have two design options: Either the CPU contains buffers, which have enough capacity to drive the inductance and capacitance inevitably associated with external memory... except the power consumption/heat dissipation problems of the CPU would increase dramatically. Or we could place high-capacity buffers only on the data bus. The second is the more sensible option.

   Having placed out-going buffers on the data bus, we are thus obliged to place incoming-buffers to enable us to *read* from memory. (Given that signals on the external data bus would be noisy, it would be a good idea to uses Schmitt trigger devices as the incoming buffers to clean-up the read signals.

3. The op-code is read from external (program) memory and forms the input to the finite-state machine, which comprises the control path. Unless we store the op-code somewhere, the value will disappear and the input to the control path will change, giving unpredictable operation. We thus need to store the op-code.

4. Providing an input to the program counter allows us to load a new value. We would want to do this in order to implement a jump instruction. Loading a new value into the PC would cause the next instruction to be fetched from the new address.

EEE339/336 Solution Sheet 3

5. An instruction fetch always consists of the same sequence of events, those required for a memory access. This results in fetch phases of the same duration. The duration of the execution phase depends on the complexity of the instruction being executed. For example, a simple instruction may execute with no further access to memory required but a complex instruction may require data to be retrieved from and written to external memory, taking much longer.

6. Microcode is a sequence of hardware-level control words, which are used to implement a higher-level machine code instruction. The microcode sequence is generally held in ROM. The ROM outputs are connected to the control inputs of components in the CPU. The op-code of the current instruction selects the start of the sequence of microcode to execute. It can be implemented in horizontal and vertical forms (see lecture notes).

   Relative to hardwired control, microcode is easier to design and modify, being particularly suited to implementing complex instructions. It is slower than hardwired control and can be inefficient if complex instructions are not used very often.

7. Hard-wired control refers to controlling the internal CPU components using a finite state machine consisting of combinational and sequential circuitry. This is circuitry is fixed and can only be modified by a change in hardware.

   Hard-wired control is generally more efficient and faster than micro-programmed control but difficult to change.

8. Hard-wired control would be preferred in a RISC machines due to its faster operation. RISC aims to have a small number of simple instructions, which can be executed in a single cycle. Hardwired control provides the fastest single cycle operation.