Multiple cycle implementation – Control

Control is more complex than in the single cycle implementation since:

- Need to define control signals for each step
- Need to know which step we are on

Two methods for designing the control unit

- Finite state machines – Hardwired control
- Microprogramming (see later)

What are the control signals needed (cf. Fig 5.32)

We first look at the control signals that will be needed at each step.

Some of these signals are for reading/writing memory, some are for writing registers, and some are for driving the choices in multiplexors.

We also need to select which function the ALU is going to perform at each step.

Instruction fetch

- Set the Mux for memory address (to have PC as input)
  IorD = 0
- Set *Memread* signal
- Set *IRwrite* signal (note that we do not need a selector for choosing between IR and MDR – Why?)
- Set The Mux for input 1 of ALU to be "from PC"
  ALUSrcA = 0
- Set the Mux for input 2 of ALU to be "from constant 4"
  ALUSrcB = 01
- Set ALUcontrol to "+"
  ALUOp = 00
- Set the Mux for input to PC to be "from ALU"
  PCSource = 00
- Set *PCwrite*

Note that the result of the addition is also stored in ALUout but, as we shall see, this is of no consequence since ALUout is going to be written each time we use the ALU.

Instruction decode and read source register(s)

Only need to select inputs for ALU (to compute branch target address), respectively "from PC"
$ALUSrcA = 0$
and "from immediate, sign-extended, shift left 2"
$ALUSrcB = 11$
and tell the ALU to do a "+"
$ALUOp = 00$

Note that the writing of temporary registers A and B does not need to be controlled. It can happen at every cycle without problem since contents of register file and of IR are not changed. When we need the values of A and B, the muxes controlling the sources to the ALU will be set appropriately.

Concept of “state”

During the first and second cycles, the signals sent by the control unit where independent of the instruction being executed.

During the 3rd cycle, though, the opcode is going to direct:

- What control lines to assert (it will be different for a load, an add, a branch etc.)
- What is to be done at the next cycle (could be access memory, write a register, start fetching a new instruction etc.)

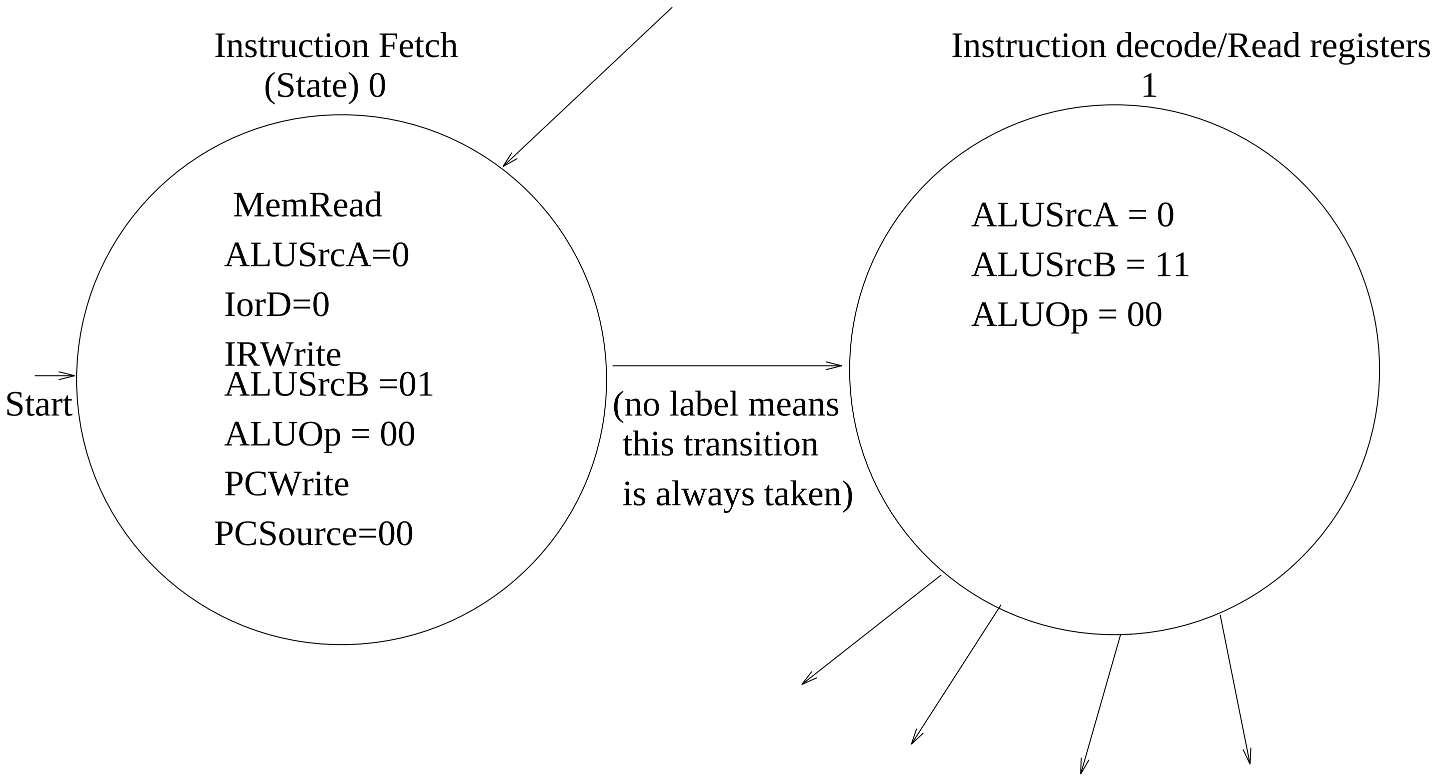
Thus at each cycle, the control unit is going to be put in a given *state*. The opcode and the current state will define the transition to the next state.

The control unit is implemented as a *finite-state machine* (cf. CSE 370, CSE 322).

The first two states

Since the data flow is the same for all instructions in the Instruction fetch cycle, there is only one state, *State 0*, associated with that cycle.

When the control is in State 0, the next state is always the unique state, say *State 1*, in the Instruction decode - Read register cycle The notation that we will use is as below:



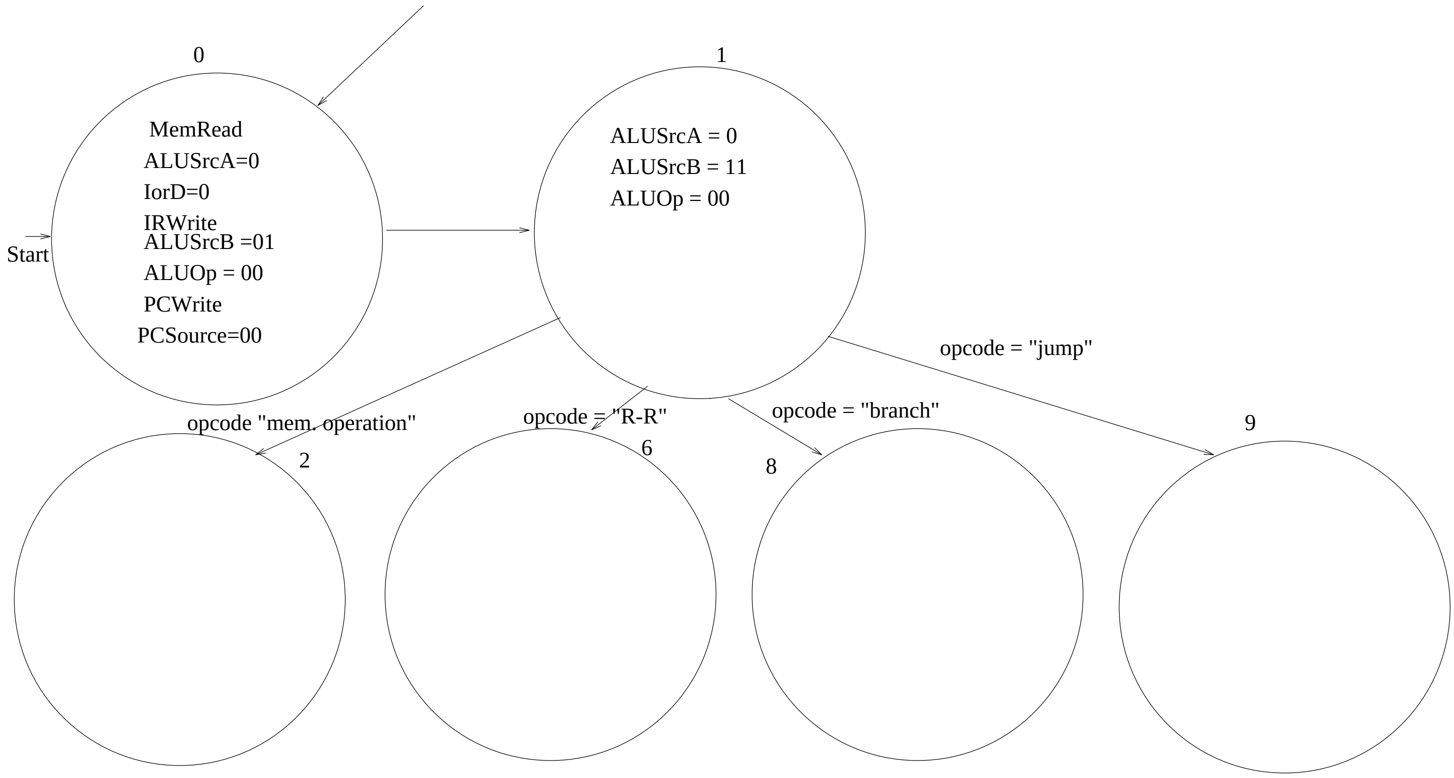
Transition from State 1

After the instruction decode / register read cycle, the data flow will depend on the type of instruction. In the subset of the ISA we look at, we have 4 types:

1. Load-Store (still need to compute the address and to access the memory, and in the case of a load to write a register)
2. Register-Register (still need to compute a result and write it in a register)
3. Branch (still need to compute the result of the condition and if need be change the PC)
4. Unconditional jump (still need to change the PC)

The transition will be dictated by the opcode

Finite State Machine – Transition from State 1



State 2: Memory address computation

Set the inputs to the ALU and the ALUOp, namely

ALUSrcA = *1* (A register)

ALUSrcB = *10* (immediate field sign extended)

ALUOp = *0* (“+”)

Transition from State 2 depends on whether we have a Load (State 3) or a Store (State 5)

State 5: Memory write (corresponds to a store operation)

The signals are:

IorD = 1 (the address of where to write comes from ALUOut)

Set *Memwrite*

Note that what to write has been sitting in Register B. It was put there in the Instruction decode cycle and written again during the memory address calculation cycle but since what was in IR and the register file has not been modified, it's the same value.

Since the instruction is completed, we always transfer from State 5 back to State 0, i.e., we are going to fetch the next instruction.

Multiple cycle implementation: the whole story

Data path with control lines: Figure 5.33

Control Unit – Finite State machine: 5.42

Note that some instructions take 3 cycles, some take 4, some take 5.

Hardwired implementation of the control unit

In single cycle implementation, we had;

Input (Opcode + function bits) $\Rightarrow$ Combinational circuit $\Rightarrow$ Output signals (data path)

Now we need to add State, i.e., a register that contains the current state. Then:

Input (Opcode + function bits + Current state) $\Rightarrow$ Combinational circuit $\Rightarrow$ Output signals (data path + setting next state)