Outline

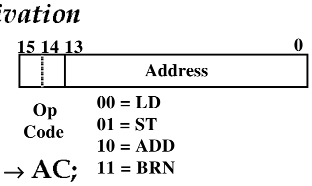
- ❍ Last time:
  - ➔ Introduction to Computer Organization
  - ➔ Control
  - ➔ Datapath
  - ➔ I/O Interface
  - ➔ Bussing Strategies
- ❍ This lecture:
  - ➔ Deriving the State Diagram & Datapath (Cont.)
  - ➔ Mapping the Datapath onto Control

Finite State Machines for Simple CPUs

State Diagram and Datapath Derivation

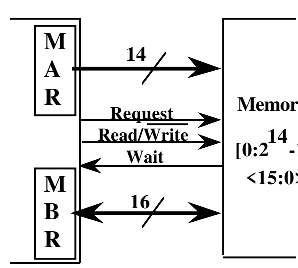
Processor Specification:

Instruction Format:



Load from memory: Mem[XXX] → AC;
Store to memory: AC → Mem[XXX];
Add from memory: AC + Mem[XXX] → AC;
Branch if accumulator is negative: AC < 0 ⇒ XXX → PC;

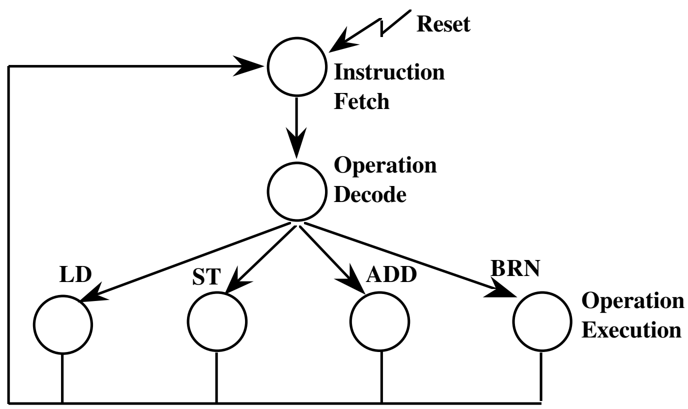
Memory Interface:



Finite State Machines for Simple CPUs

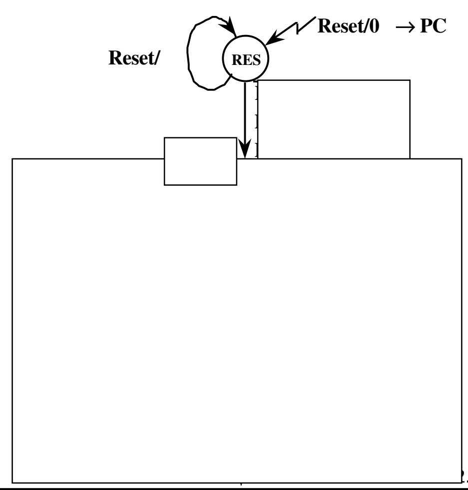
Deriving the State Diagram and Datapath

First pass state diagram:



Deriving the State Diagram and Datapath

Assume Synchronous Mealy Machine:
Transitions associated with arcs rather than states



Reset State (State 0) and Instruction Fetch Sequence

On Reset:
- **zero the PC**
- **Mem Request unasserted**
- **Mem asserts Wait signal**

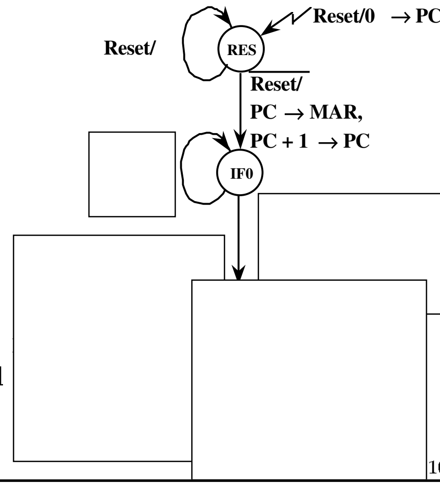
Deriving the State Diagram and Datapath

Assume Synchronous Mealy Machine:
Transitions associated with arcs rather than states

Reset State (State 0) and Instruction Fetch Sequence

On Reset:
- **zero the PC**
- **Mem Request unasserted**
- **Mem asserts Wait signal**

Instruction Fetch:
- **issue read request**
- **4 cycle handshake on Wait signal**



Deriving the State Diagram and Datapath

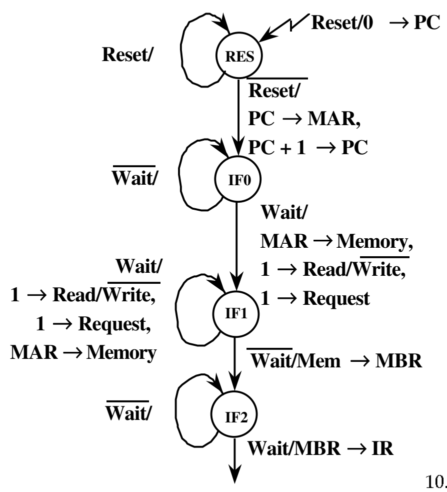
Assume Synchronous Mealy Machine:
Transitions associated with arcs rather than states

Reset State (State 0) and Instruction Fetch Sequence

On Reset:
- zero the PC
- Mem Request unasserted
- Mem asserts Wait signal

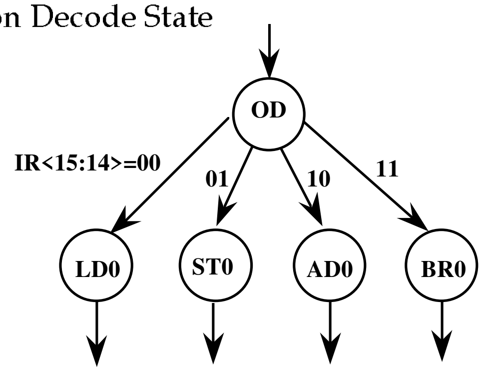
Instruction Fetch:
- issue read request
- 4 cycle handshake on Wait signal

Note: No explicit mention of the busses being used to implement register transfers!



Deriving the State Diagram and Datapath

Operation Decode State



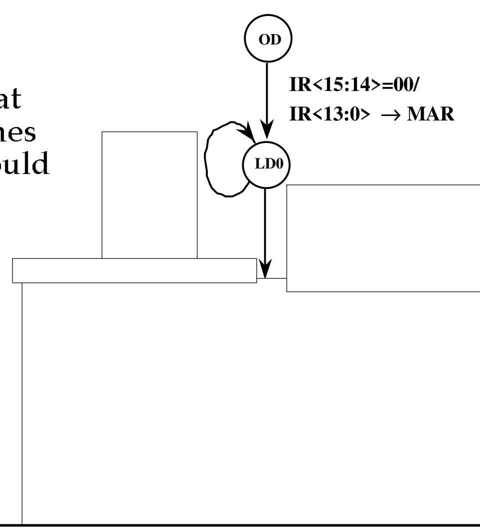
Four Way Next State Branch based on opcode bits

Deriving the State Diagram and Datapath

Execution Sequences

Load Sequence

like IFetch, except that operand address comes from IR and data should be loaded into AC

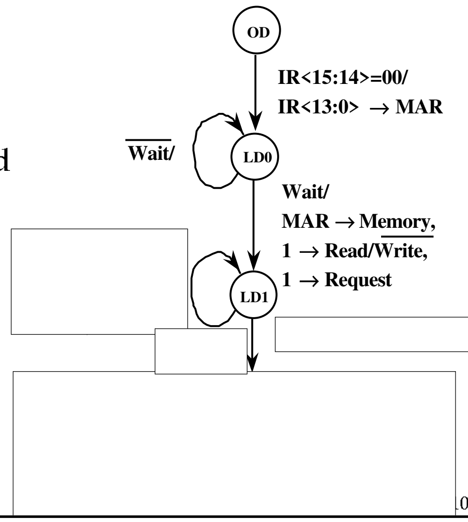


Deriving the State Diagram and Datapath

Execution Sequences

Load Sequence

like IFetch, except that operand address comes from IR and data should be loaded into AC

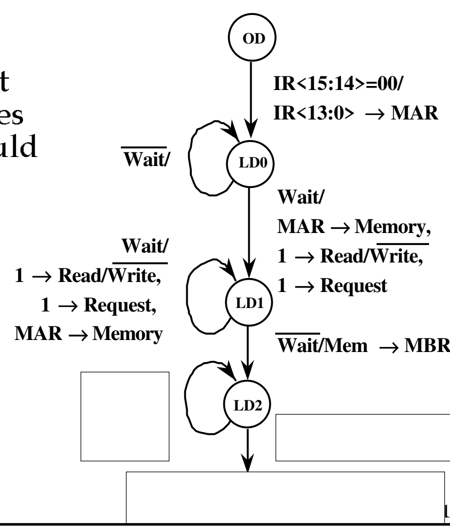


Deriving the State Diagram and Datapath

Execution Sequences

Load Sequence

like IFetch, except that operand address comes from IR and data should be loaded into AC

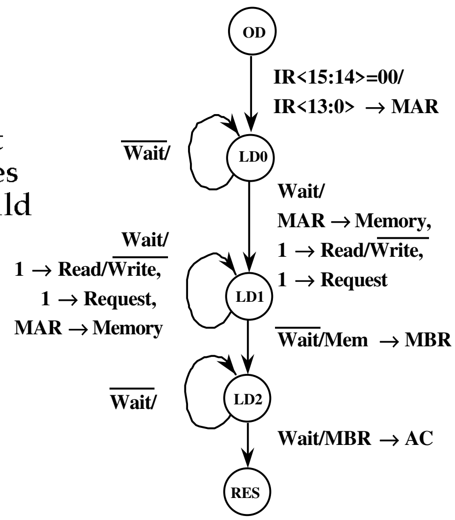


Deriving the State Diagram and Datapath

Execution Sequences

Load Sequence

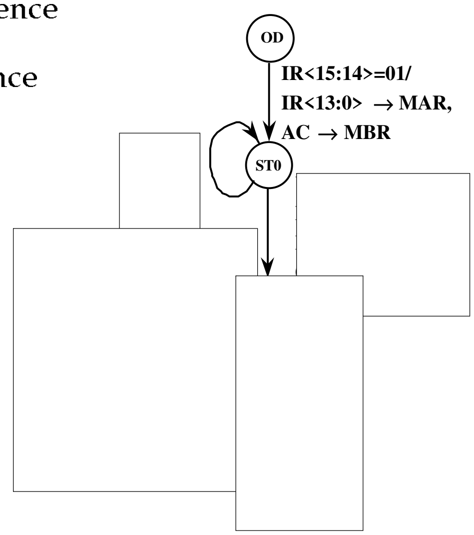
like IFetch, except that operand address comes from IR and data should be loaded into AC



Deriving the State Diagram and Datapath

Store Execution Sequence

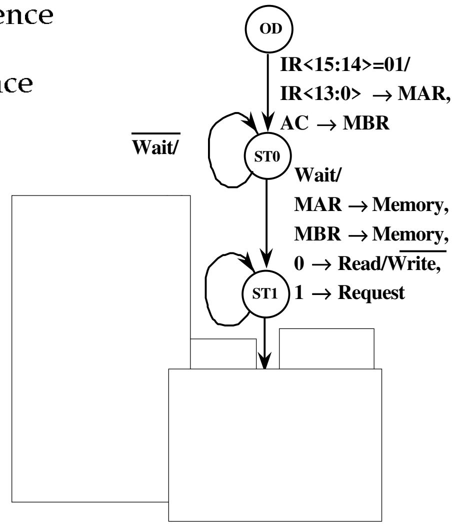
Memory write sequence



Deriving the State Diagram and Datapath

Store Execution Sequence

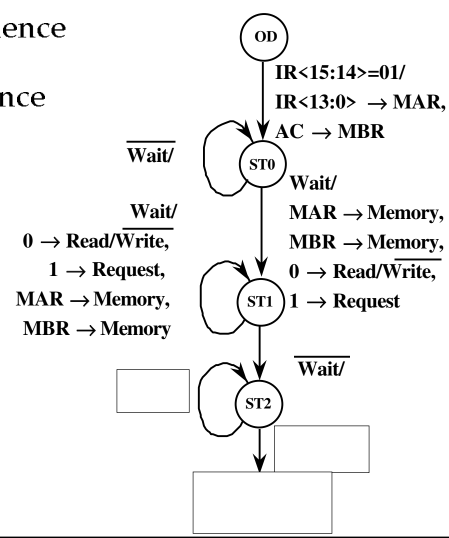
Memory write sequence



Deriving the State Diagram and Datapath

Store Execution Sequence

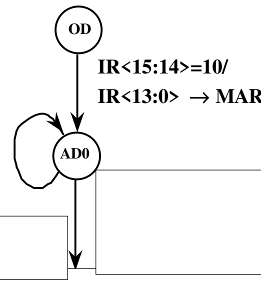
Memory write sequence



Deriving the State Diagram and Datapath

Add Execution Sequence

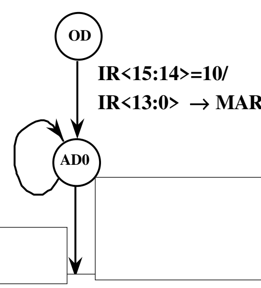
Similar to Load sequence
Add MBR, AC rather than
simply transfer MBR to AC



Deriving the State Diagram and Datapath

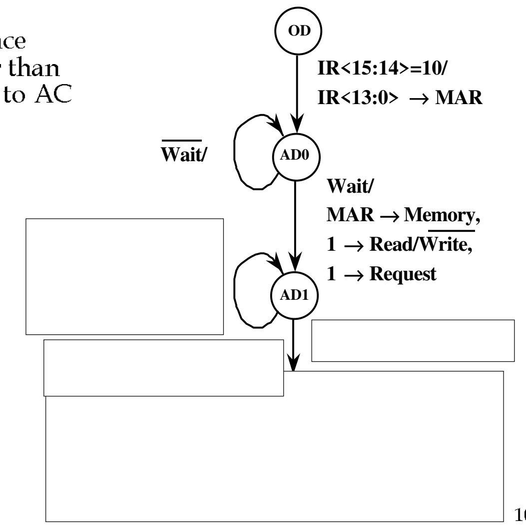
Add Execution Sequence

Similar to Load sequence
Add MBR, AC rather than
simply transfer MBR to AC



Deriving the State Diagram and Datapath

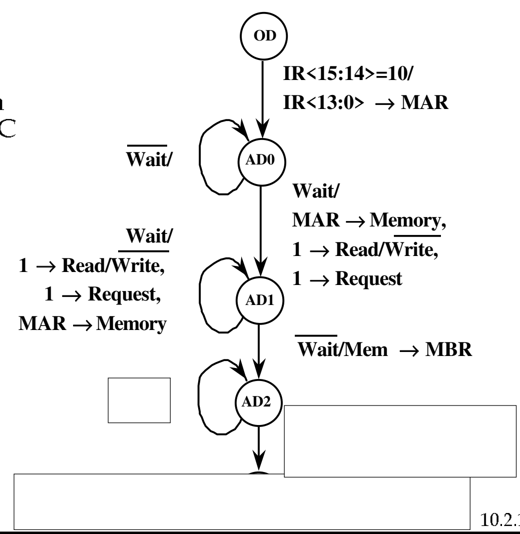
Similar to Load sequence
Add MBR, AC rather than
simply transfer MBR to AC



Deriving the State Diagram and Datapath

Add Execution Sequence

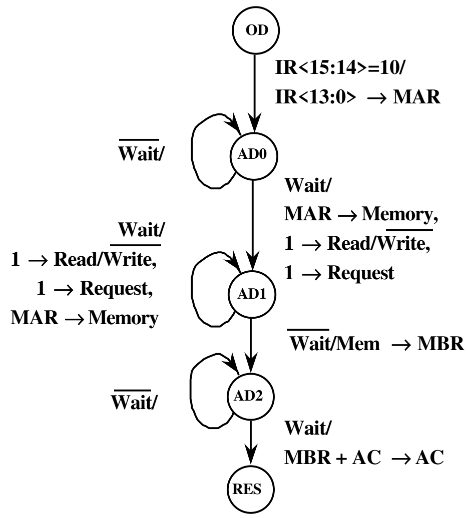
Similar to Load sequence
Add MBR, AC rather than
simply transfer MBR to AC



Deriving the State Diagram and Datapath

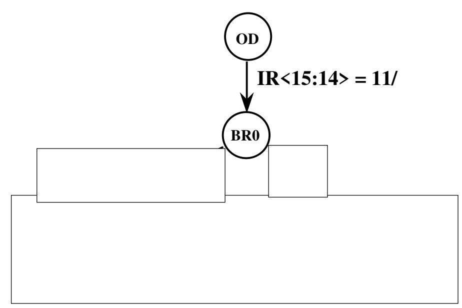
Add Execution Sequence

Similar to Load sequence
Add MBR, AC rather than simply transfer MBR to AC



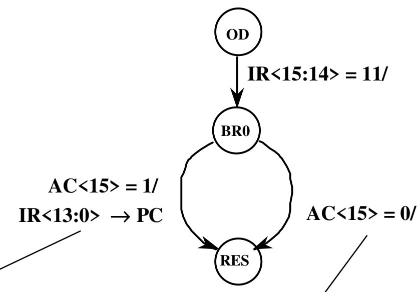
Deriving the State Diagram and Datapath

Branch Execution Sequence



Deriving the State Diagram and Datapath

Branch Execution Sequence



Replace PC with Operand Address if AC < 0

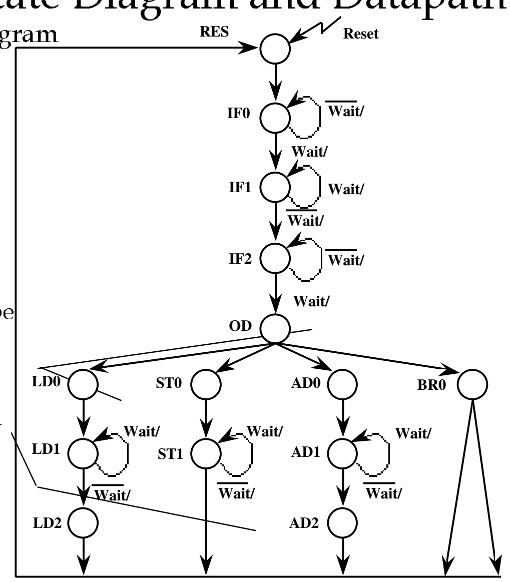
Otherwise, do nothing

Deriving the State Diagram and Datapath

Revised/Complete State Diagram

Simplify Wait Looping

Eliminate some Wait states



At this point, Wait must be asserted, so why loop on Wait?

Why $\overline{\text{loop}}$ on $\overline{\text{Wait}}$ when resync will take place at state IF0?

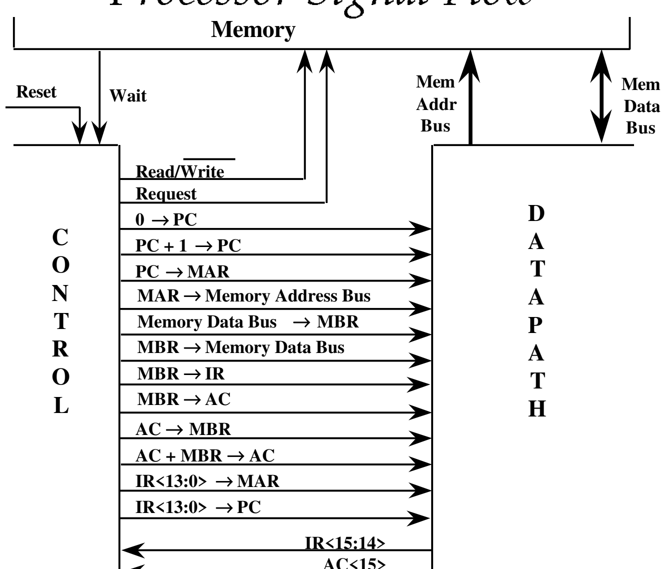
Deriving the State Diagram and Datapath

State Machine Inputs and Outputs so far:

Inputs:
- Reset
- Wait
- IR<15:14>
- AC<15>

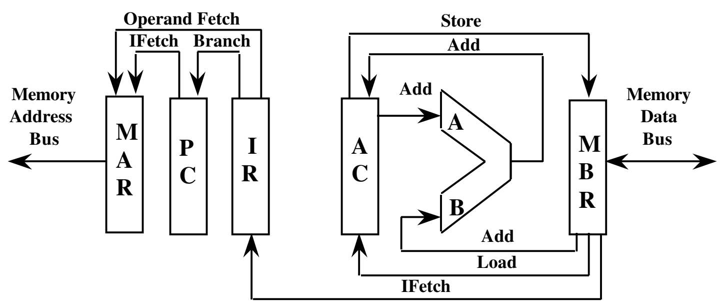
Outputs:
- 0 → PC
- PC + 1 → PC
- PC → MAR
- MAR → Memory Address Bus
- Memory Data Bus → MBR
- MBR → Memory Data Bus
- MBR → IR
- MBR → AC
- AC → MBR
- AC + MBR → AC
- IR<13:0> → MAR
- IR<13:0> → PC
- 1 → Read/Write
- 0 → Read/Write
- 1 → Request

Processor Signal Flow



Mapping onto Datapath Control

Specification so far is independent of bussing strategy
Implied transfers:



This is the point-to-point connection scheme

Mapping onto Datapath Control

Observe that instruction fetch and operand fetch take place at different times

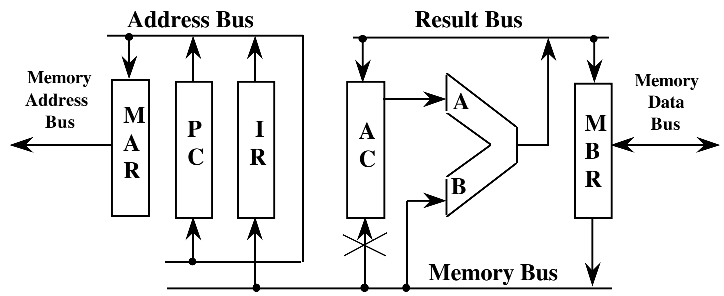
This implies that IR, PC, and MAR transfers can be implemented by single bus (Address Bus)

Combine MBR, IR, ALU B, and AC connections (Memory Bus)

Combine ALU, AC, and MBR connections (Result Bus)

Three bus architecture:
AC + MBR → AC implemented in single state

Mapping onto Datapath Control



AC has two inputs, RBUS and MBUS
(Other registers except MBR have single input and output)

Dual ported configuration is more complex

Better idea: reuse existing paths were possible
MBR $\rightarrow$ AC transfer implemented by PASS B ALU operation

Mapping onto Datapath Control

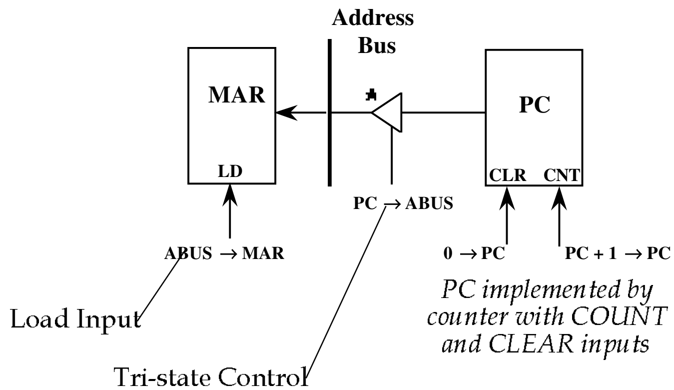
Detailed implementation of register transfer operations
More detailed control operations are called *microoperations*

One register transfer operation = several microoperations

Some operations directly implemented by functional units:
e.g., ADD, Pass B, 0 $\rightarrow$ PC, PC + 1 $\rightarrow$ PC

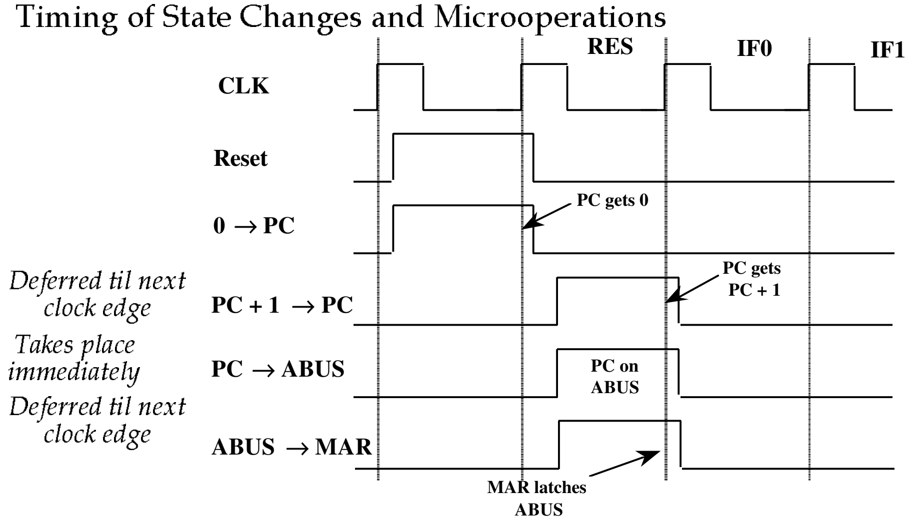
Some operations require multiple control operations:
e.g., PC $\rightarrow$ MAR implemented as
PC $\rightarrow$ ABUS and ABUS $\rightarrow$ MAR

Mapping onto Datapath Control



Mapping onto Datapath Control

Timing of State Changes and Microoperations



Mapping onto Datapath Control

Relationship between register transfer and microoperations:

Register Transfer	Microoperations
0 → PC	0 → PC (delayed);
PC + 1 → PC	PC + 1 → PC (delayed);
PC → MAR	PC → ABUS (immediate), ABUS → MAR (delayed);
MAR → Address Bus	MAR → Address Bus (immediate);
Data Bus → MBR	Data Bus → MBR (delayed);
MBR → Data Bus	MBR → Data Bus (immediate);
MBR → IR	MBR → ABUS (immediate), ABUS → IR (delayed);
MBR → AC	MBR → MBUS (immediate), MBUS → ALU B (immediate), ALU PASS B (immediate), ALU Result → RBUS (immediate), RBUS → AC (delayed);

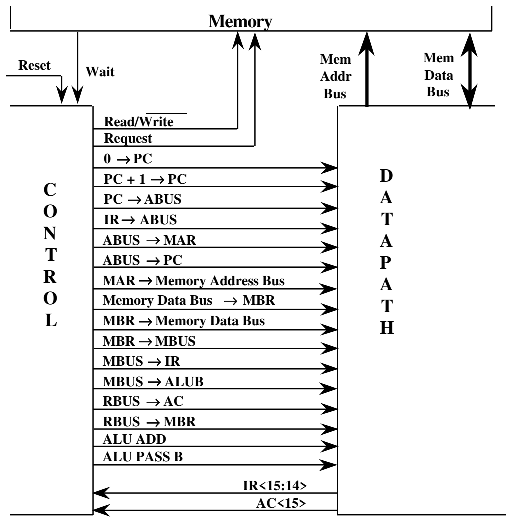
Mapping onto Datapath Control

Relationship between register transfer and microoperations:

Register Transfer	Microoperations
AC → MBR	AC → RBUS (immediate), RBUS → MBR (delayed);
AC + MBR → AC	AC → ALU A (immediate), MBR → MBUS (immediate), MBUS → ALU B (immediate), ALU ADD (immediate), ALU Result → RBUS (immediate), RBUS → AC (delayed);
IR<13:0> → MAR	IR → ABUS (immediate), ABUS → IR (delayed);
IR<13:0> → PC	IR → ABUS (immediate), ABUS → PC (delayed);
1 → Read/Write	Read (immediate);
0 → Read/Write	Write (immediate);
1 → Request	Request (immediate);

Special microoperations for AC → ALU and ALU Result → RBUS not strictly necessary since these connections can be hardwired

Mapping onto Datapath Control



Revised microoperation signal flow

5 inputs

make sure that Reset and Wait are synchronized

16 datapath control lines

2 memory control lines