Simple Instruction Pipelining

Krste Asanovic
Laboratory for Computer Science
M.I.T.

http://www.csg.lcs.mit.edu/6.823

Processor Performance Equation

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

- **Instructions per program depends on source code, compiler technology, and ISA**
- **Microcoded DLX from last lecture had cycles per instruction (CPI) of around 7 *minimum***
- **Time per cycle for microcoded DLX fixed by microcode cycle time**

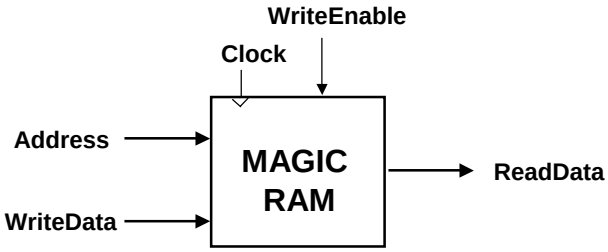
  → **mostly ROM access + next μPC select logic**

Pipelined DLX

To pipeline DLX:

- **First build unpipelined DLX with CPI=1**
- **Next, add pipeline registers to reduce cycle time while maintaining CPI=1**

A Simple Memory Model

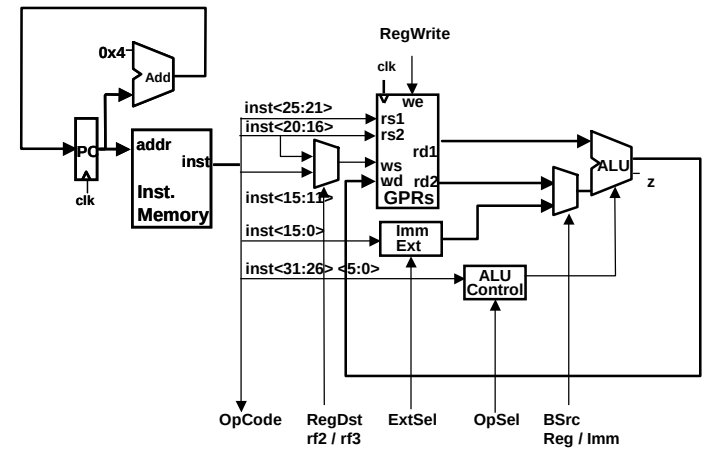


Reads and writes are always completed in one cycle

- **a Read can be done any time (i.e. combinational)**
- **a Write is performed at the rising clock edge if it is enabled**

$\Rightarrow$ ***the write address and data must be stable at the clock edge***

Datapath for ALU Instructions



6	5	5	5	5	6	
0	rf1	rf2	rf3	0	func	rf3 ← (rf1) func (rf2)
opcode	rf1	rf2	immediate			rf2 ← (rf1) op immediate

Datapath for Memory Instructions

Should program and data memory be separate?

Harvard style: ***separate***
(Aiken and Mark 1 influence)

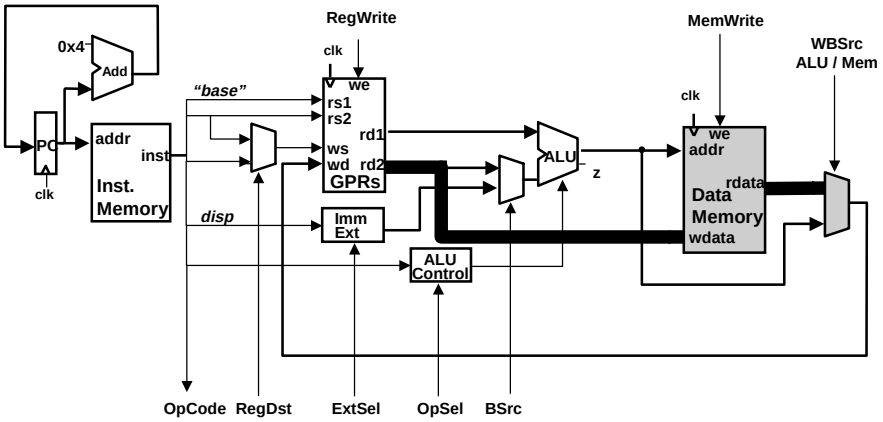
- **read-only program memory**
- **read/write data memory**
  at some level the two memories have to be the same

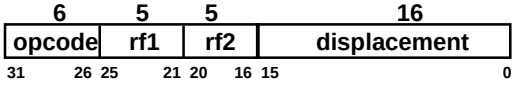
Princeton style: ***the same***
(von Neumann's influence)

- **A Load or Store instruction requires accessing the memory more than once during its execution**

Load/Store Instructions:

Harvard-Style Datapath

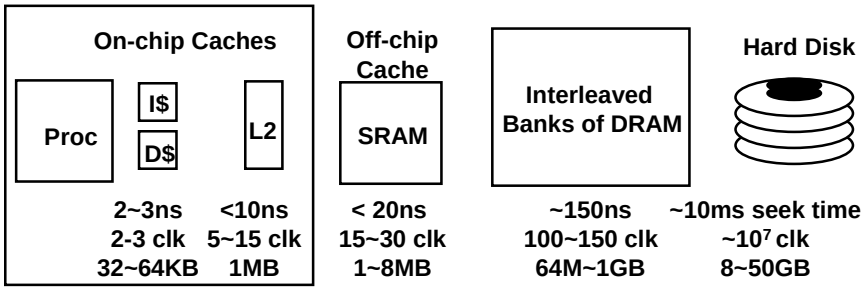




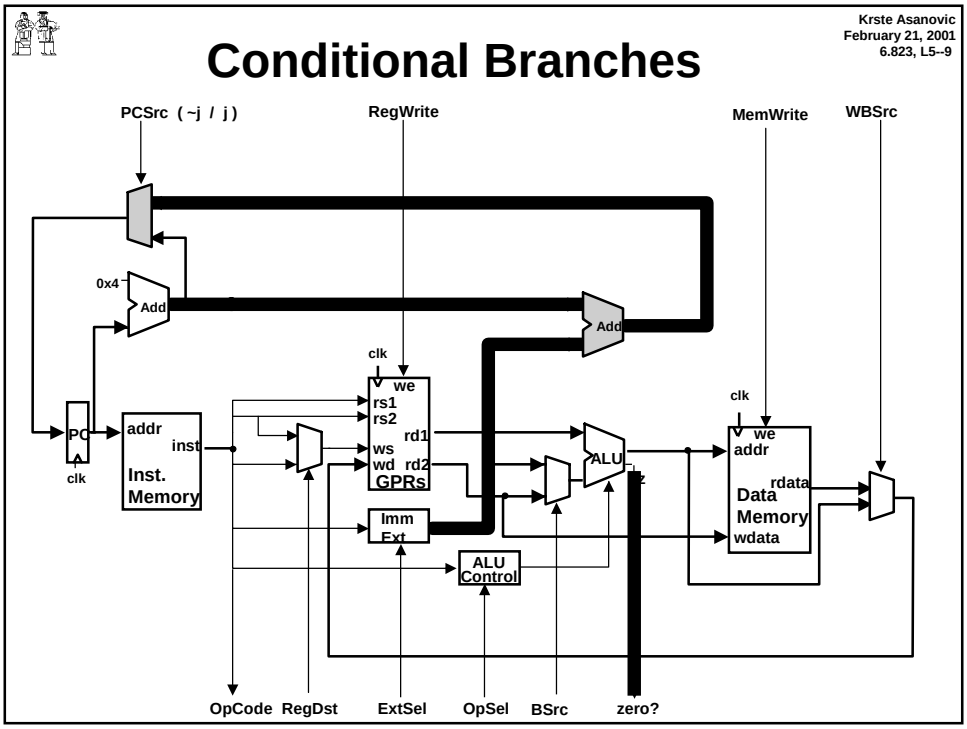
addressing mode
(rf1) + displacement

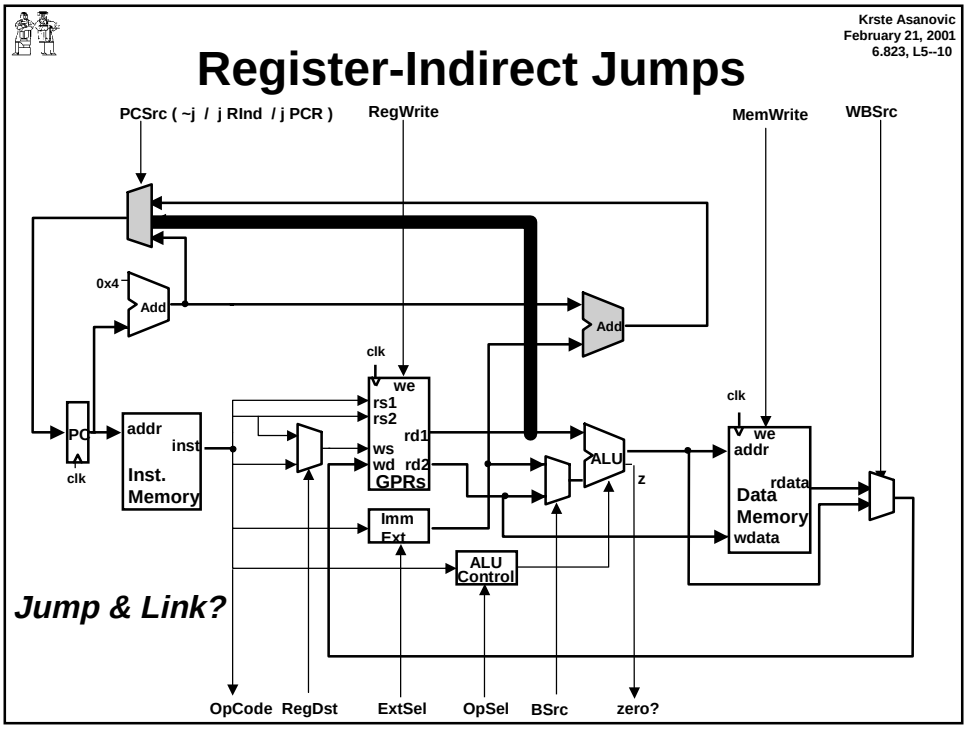
rf1 is the base register
rf2 is the destination of a Load or the source for a Store

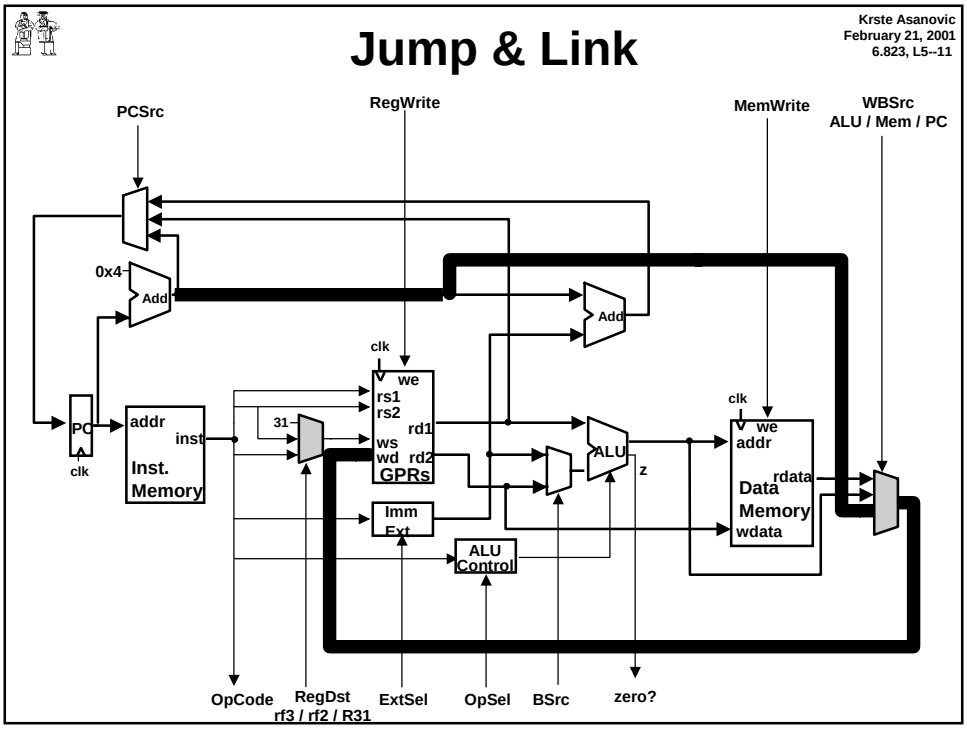
Memory Hierarchy *c.*2000

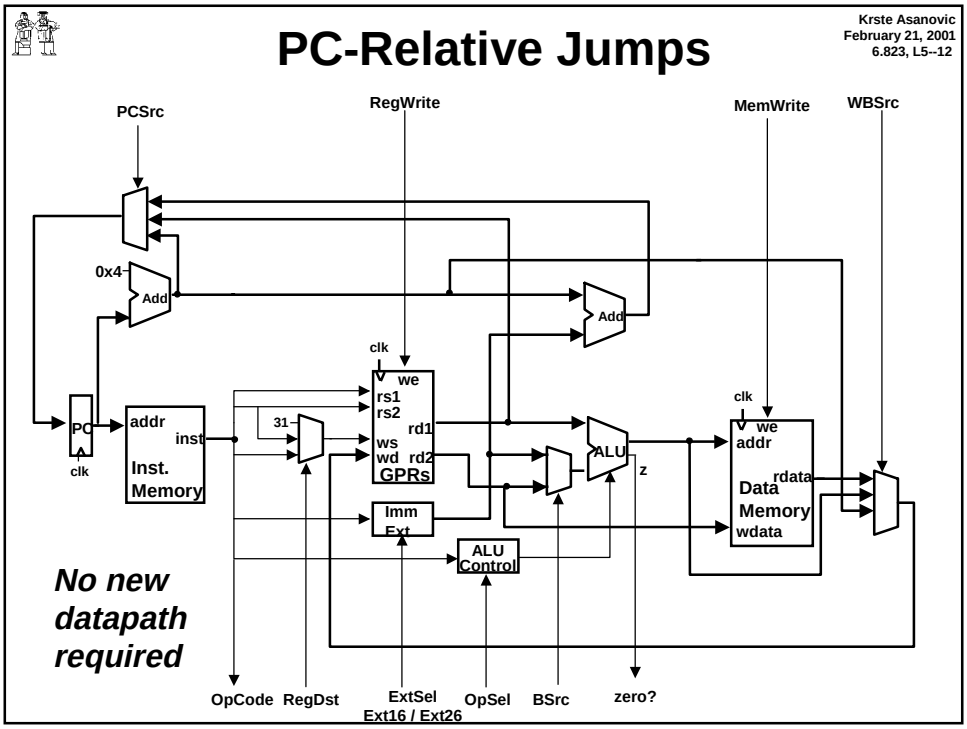


Our memory model is a good approximation of the hierarchical memory system when we hit in the on-chip cache


Krste Asanovic
February 21, 2001
6.823, L5--9
Conditional Branches
PCSrc (~j / j)
RegWrite
MemWrite
WBSrc
0x4
Add
Add
clk
we
rs1
rs2
rd1
ws
wd rd2
GPRs
Imm
Ext
PC
clk
addr
inst
Inst.
Memory
ALU
z
ALU
Control
clk
we
addr
rdata
Data
Memory
wdata
OpCode
RegDst
ExtSel
OpSel
BSrc
zero?


Krste Asanovic
February 21, 2001
6.823, L5--10
Register-Indirect Jumps
PCSrc (~j / j RInd / j PCR)
RegWrite
MemWrite
WBSrc
0x4
Add
Add
clk
we
rs1
rs2
rd1
ws
wd rd2
GPRs
Imm
Ext
PC
clk
addr
inst
Inst.
Memory
ALU
z
ALU
Control
clk
we
addr
rdata
Data
Memory
wdata
Jump & Link?
OpCode
RegDst
ExtSel
OpSel
BSrc
zero?


Jump & Link
Krste Asanovic
February 21, 2001
6.823, L5--11
PCSrc
RegWrite
MemWrite
WBSrc
ALU / Mem / PC
0x4
Add
Add
clk
we
rs1
rs2
rd1
ws
wd
rd2
GPRs
PC
clk
addr
inst
Inst.
Memory
31
ALU
z
clk
we
addr
rdata
Data
Memory
wdata
Imm
Ext
ALU
Control
OpCode
RegDst
rf3 / rf2 / R31
ExtSel
OpSel
BSrc
zero?


PC-Relative Jumps
Krste Asanovic
February 21, 2001
6.823, L5--12
PCSrc
RegWrite
MemWrite
WBSrc
0x4
Add
Add
clk
we
rs1
rs2
rd1
ws
wd
rd2
GPRs
PC
clk
addr
inst
Inst.
Memory
31
ALU
z
clk
we
addr
rdata
Data
Memory
wdata
Imm
Ext
ALU
Control
No new
datapath
required
OpCode
RegDst
ExtSel
Ext16 / Ext26
OpSel
BSrc
zero?

Single-Cycle Hardwired Control

We will assume

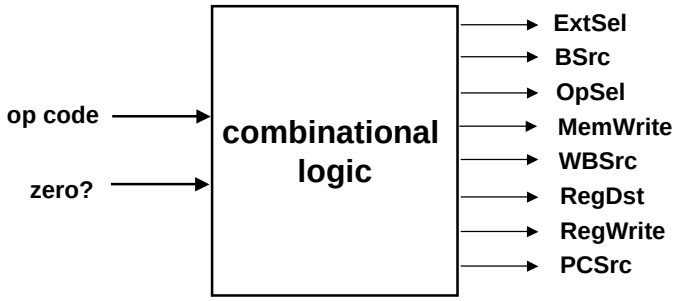
- **clock period is sufficiently long for all of the following steps to be "completed":**

  1. **instruction fetch**
  2. **decode and register fetch**
  3. **ALU operation**
  4. **data fetch if required**
  5. **register write-back setup time**

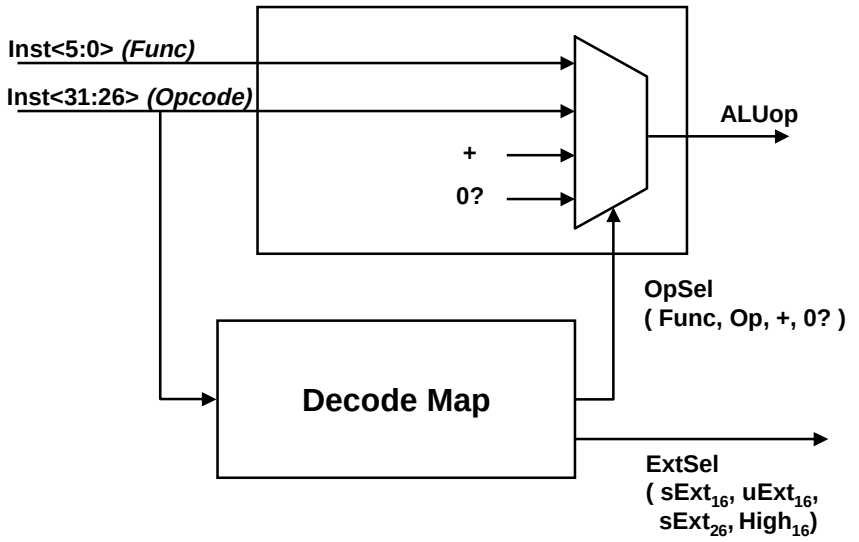
  $\Rightarrow \quad t_C > t_{IFetch} + t_{RFetch} + t_{ALU} + t_{DMem} + t_{RWB}$

- **At the rising edge of the following clock, the PC, the register file and the memory are updated**

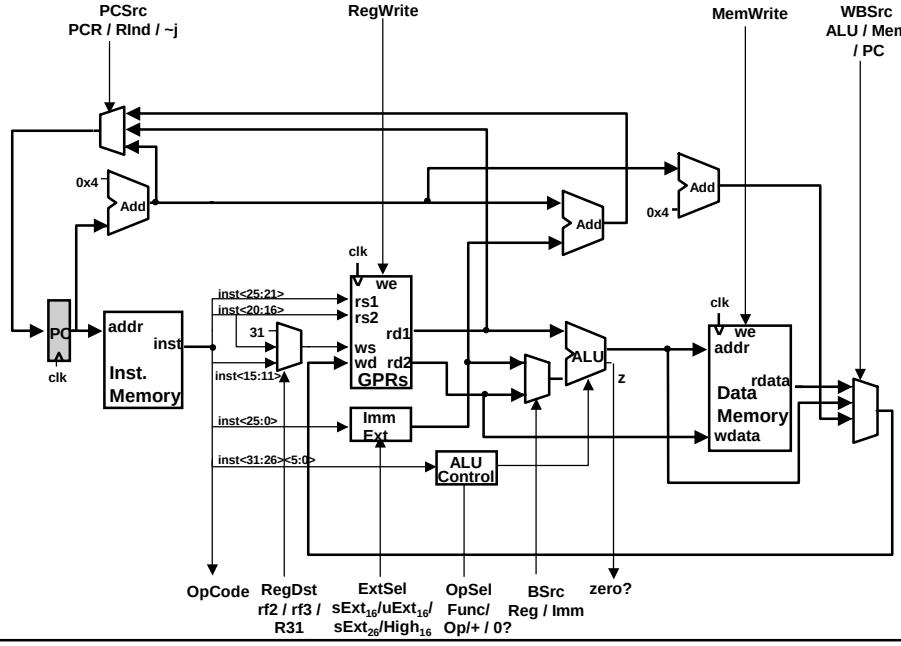
Hardwired Control is pure Combinational Logic



ALU Control & Immediate Extension



Hardwired Control *worksheet*



Hardwired Control Table

	Ext Sel	B Src	Op Sel	Mem Write	Reg Write	WB Src	Reg Dst	PC Src
ALU ALUu								
ALUi ALUui								
LW SW								
$BEQZ_{taken}$ $BEQZ_{\sim taken}$								
J JAL								
JR JALR								

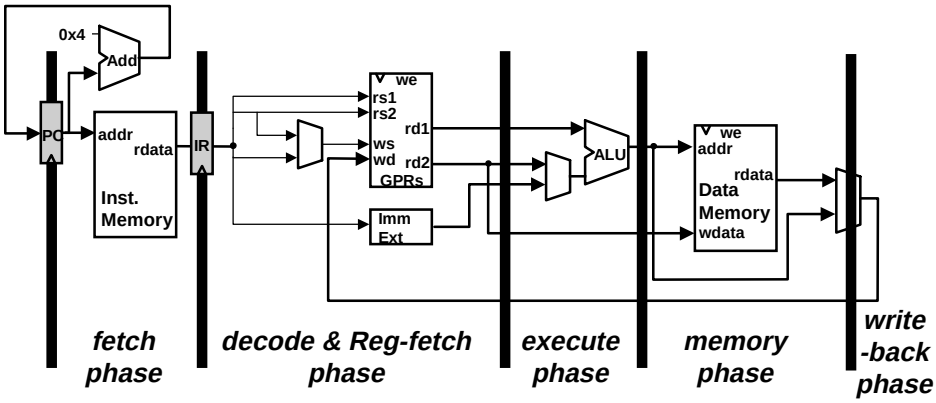
BSrc = Reg / Imm WBSrc = ALU / Mem / PC RegDst = rf2 / rf3 / R31
PCSrc = PCR/RInd / ~j

Hardwired Control Table: *Harvard DLX*

	Ext Sel	B Src	Op Sel	Mem Write	Reg Write	WB Src	Reg Dst	PC Src
ALU	*	Reg	Func	no	yes	ALU	rf3	~j
ALUu	*	Reg	Func	no	yes	ALU	rf3	~j
ALUi	$sExt_{16}$	Imm	Op	no	yes	ALU	rf2	~j
ALUui	$uExt_{16}$	Imm	Op	no	yes	ALU	rf2	~j
LW	$sExt_{16}$	Imm	+	no	yes	Mem	rf2	~j
SW	$sExt_{16}$	Imm	+	yes	no	*	*	~j
$BEQZ_{zero?=1}$	$sExt_{16}$	*	0?	no	no	*	*	PCR
$BEQZ_{zero?=0}$	$sExt_{16}$	*	0?	no	no	*	*	~j
J	$sExt_{26}$	*	*	no	no	*	*	PCR
JAL	$sExt_{26}$	*	*	no	yes	PC	R31	PCR
JR	*	*	*	no	no	*	*	RInd
JALR	*	*	*	no	yes	PC	R31	RInd

BSrc = Reg / Imm WBSrc = ALU / Mem / PC RegDst = rf2 / rf3 / R31
PCSrc1 = j / ~j PCSrc2 = PCR / RInd

Pipelined DLX Datapath



Clock period can be reduced by dividing the execution of an instruction into multiple cycles

$t_C > \max \{t_{IM}, t_{RF}, t_{ALU}, t_{DM}, t_{RW}\} = t_{DM}$ ***(probably)***

However, CPI will increase unless instructions are pipelined

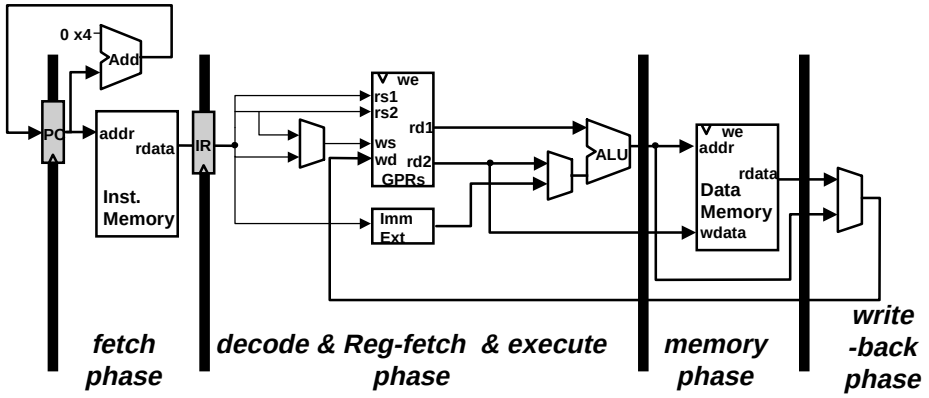
How to divide the datapath into stages

Suppose memory is significantly slower than other stages. In particular, suppose

$t_{IM} = t_{DM} = 10$ units
$t_{ALU} = 5$ units
$t_{RF} = t_{RW} = 1$ unit

Since the slowest stage determines the clock, it may be possible to combine some stages without any loss of performance

Minimizing Critical Path



$t_C > \max \{t_{IM}, t_{RF} + t_{ALU}, t_{DM}, t_{RW}\}$

Write-back stage takes much less time than other stages. Suppose we combined it with the memory phase

$\Rightarrow$ ***increase the critical path by 10%***

Maximum Speedup by Pipelining

For the 4-stage pipeline, given

$t_{IM} = t_{DM} =$ 10 units, $t_{ALU} = 5$ units, $t_{RF} = t_{RW} = 1$ unit

t_C could be reduced from 27 units to 10 units

$\Rightarrow$ **speedup = 2.7**

However, if $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW} = 5$ units
The same 4-stage pipeline can reduce t_C from 25 units to 10 units

$\Rightarrow$ **speedup = 2.5**

But, since $t_{IM} = t_{DM} = t_{ALU} = t_{RF} = t_{RW}$, it is possible to achieve higher speedup with more stages in the pipeline.

A 5-stage pipeline can reduce t_C from 25 units to 5 units

$\Rightarrow$ **speedup = 5**

Technology Assumptions

We will assume

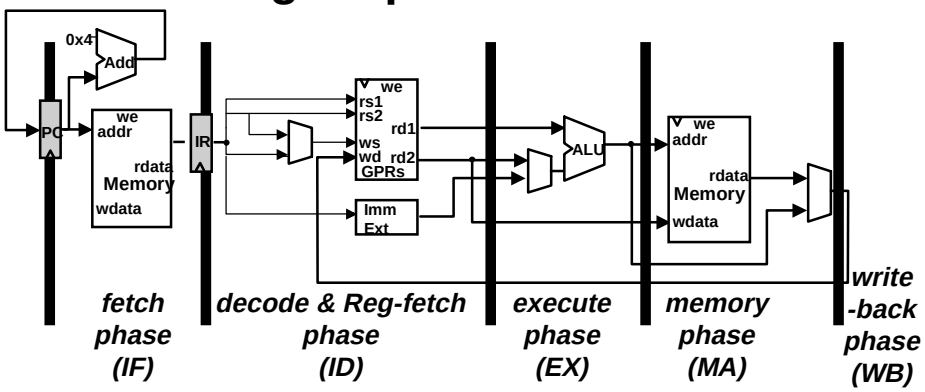
- **A small amount of very fast memory (caches) backed up by a large, slower memory**
- **Fast ALU (at least for integers)**
- **Multiported Register files (slower!).**

It makes the following timing assumption valid

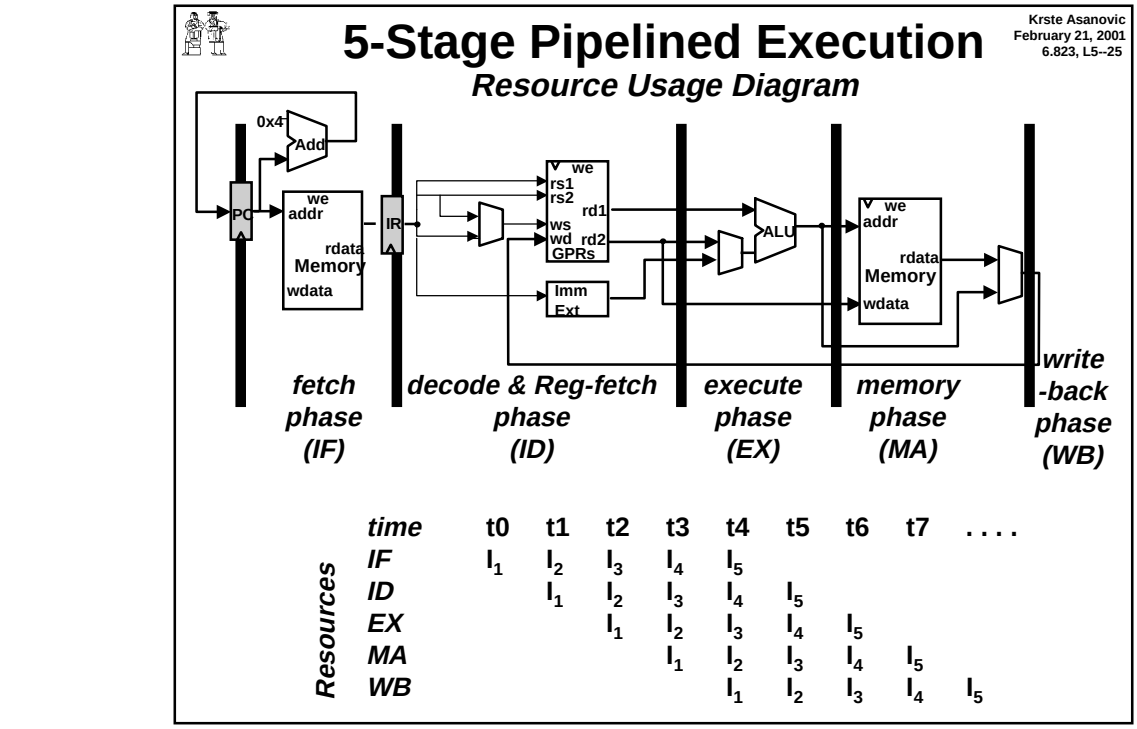
$$t_{IM} \approx t_{RF} \approx t_{ALU} \approx t_{DM} \approx t_{RW}$$

A 5-stage pipelined Harvard-style architecture will be the focus of our detailed design

5-Stage Pipelined Execution



time	t0	t1	t2	t3	t4	t5	t6	t7	. . . .
instruction1	IF_1	ID_1	EX_1	MA_1	WB_1				
instruction2		IF_2	ID_2	EX_2	MA_2	WB_2			
instruction3			IF_3	ID_3	EX_3	MA_3	WB_3		
instruction4				IF_4	ID_4	EX_4	MA_4	WB_4	
instruction5					IF_5	ID_5	EX_5	MA_5	WB_5

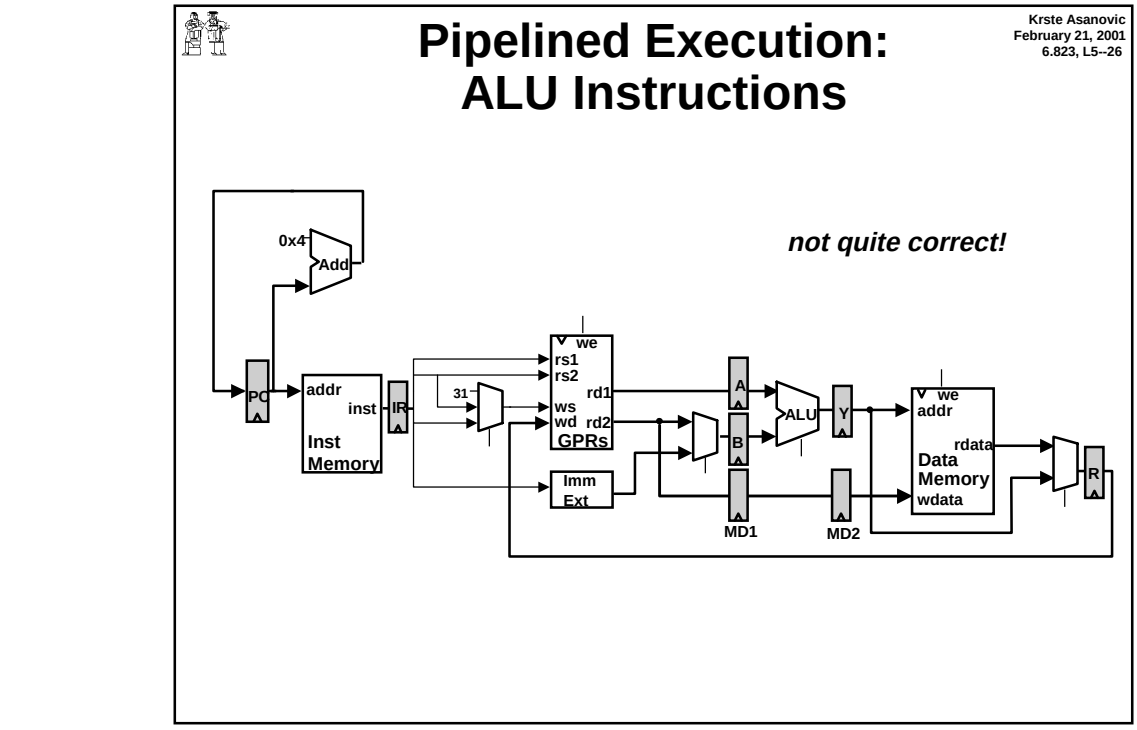


5-Stage Pipelined Execution

Krste Asanovic
February 21, 2001
6.823, L5--25

Resource Usage Diagram

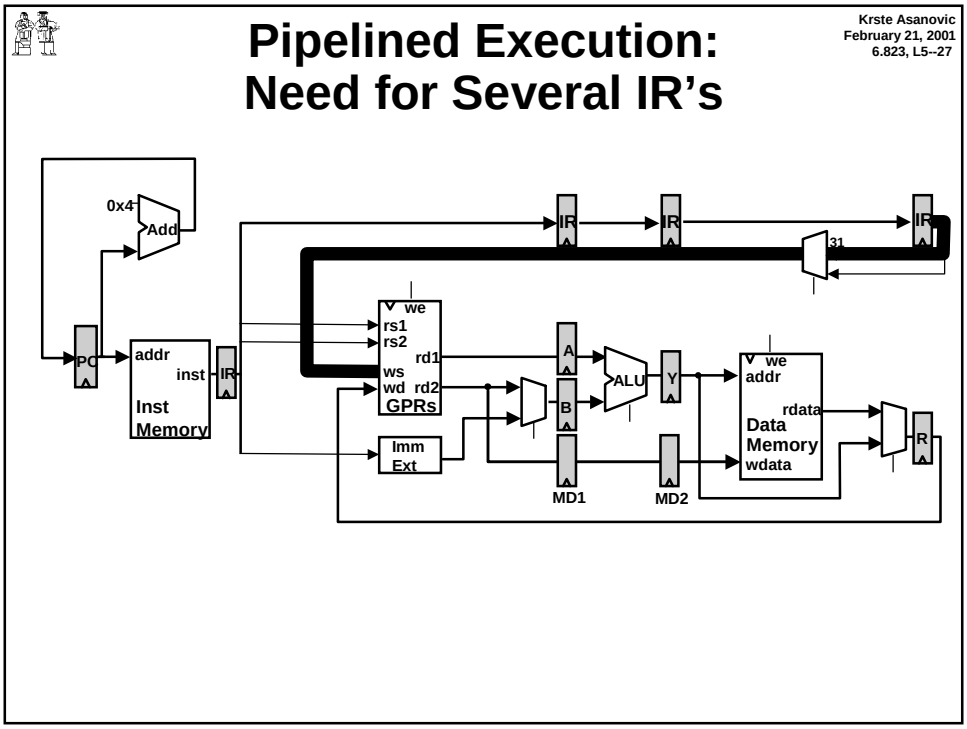
Resources	*time*	t0	t1	t2	t3	t4	t5	t6	t7	. . . .
	IF	I_1	I_2	I_3	I_4	I_5				
	ID		I_1	I_2	I_3	I_4	I_5			
	EX			I_1	I_2	I_3	I_4	I_5		
	MA				I_1	I_2	I_3	I_4	I_5	
	WB					I_1	I_2	I_3	I_4	I_5



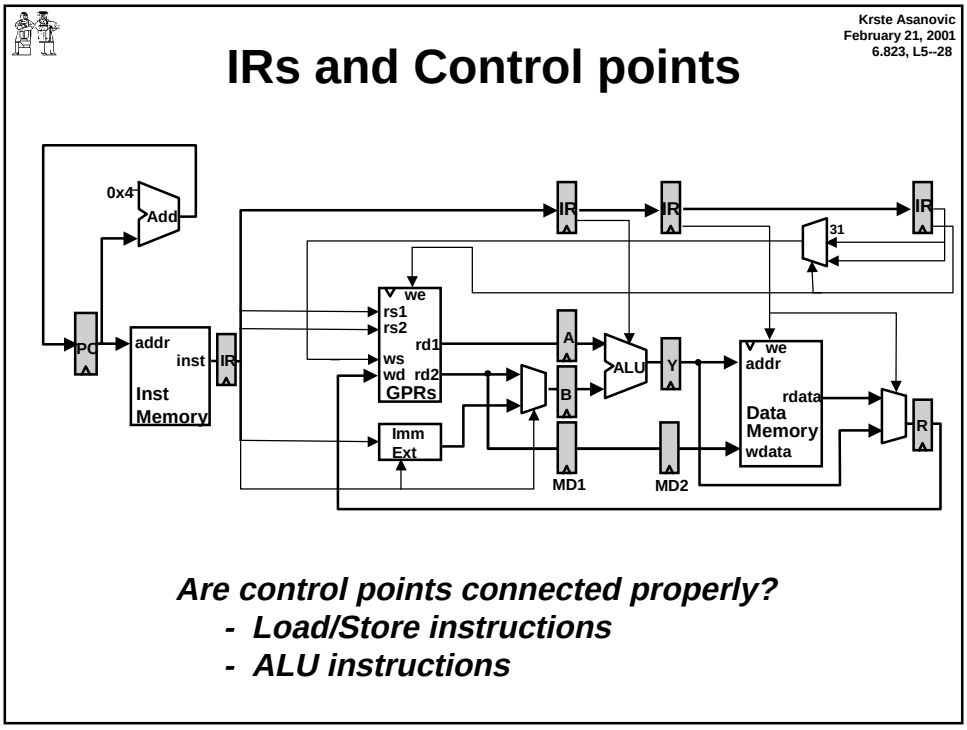
Pipelined Execution: ALU Instructions

Krste Asanovic
February 21, 2001
6.823, L5--26

not quite correct!



Pipelined Execution: Need for Several IR's



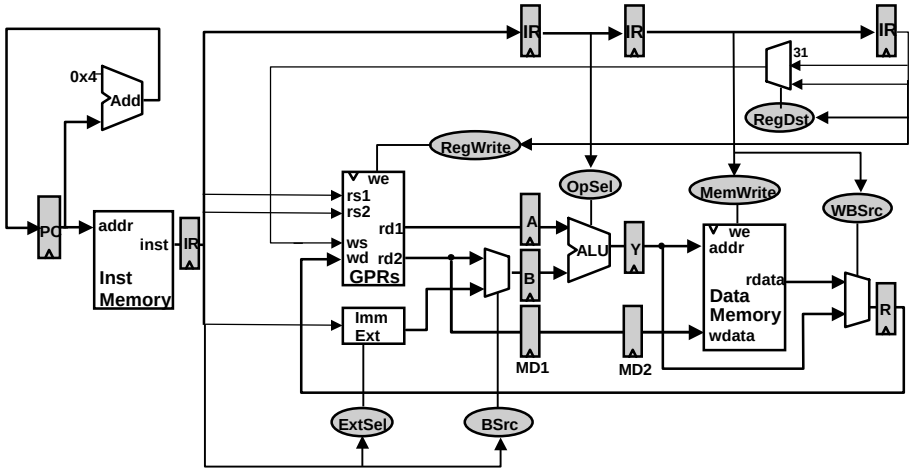
IRs and Control points

Are control points connected properly?

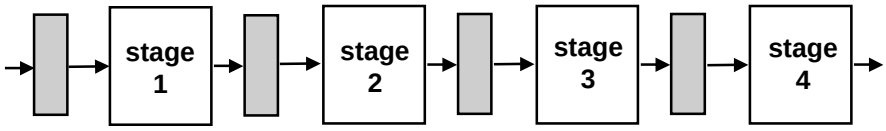
- ***Load/Store instructions***
- ***ALU instructions***

Pipelined DLX Datapath

without jumps



An Ideal Pipeline



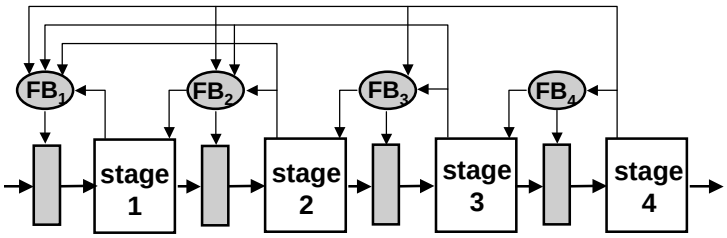
- **All objects go through the same stages**
- **No sharing of resources between any two stages**
- **Propagation delay through all pipeline stages is equal**
- **The scheduling of an object entering the pipeline is not affected by the objects in other stages**

  These conditions generally hold for industrial assembly lines. An instruction pipeline, however, cannot satisfy the last condition. Why?

How Instructions can Interact with each other in a pipeline

- An instruction in the pipeline may need a resource being used by another instruction in the pipeline
  structural hazard
- An instruction may produce data that is needed by a later instruction
  data hazard
- In the extreme case, an instruction may determine the next instruction to be executed
  control hazard (branches, interrupts,...)

Feedback to Resolve Hazards



Controlling pipeline in this manner works provided ***the instruction at stage i+1 can complete without any interference from instructions in stages 1 to i*** (otherwise deadlocks may occur)

Feedback to previous stages is used to ***stall or kill instructions***