CS252
Graduate Computer Architecture
Lecture 2

Review of Instruction Sets, Pipelines, and Caches

Prof. David Culler
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~culler/courses/cs252-s05

Review, #1

- Technology is changing rapidly:

	Capacity	Speed
Logic	2x in 3 years	2x in 3 years
DRAM	4x in 3 years	2x in 10 years
Disk	4x in 3 years	2x in 10 years
Processor	(n.a.)	2x in 1.5 years

- What was true five years ago is not necessarily true now.
- Execution time is the REAL measure of computer performance!
  - Not clock rate, not CPI
- "X is n times faster than Y" means:

$$\frac{\text{ExTime(Y)}}{\text{ExTime(X)}} = \frac{\text{Performance(X)}}{\text{Performance(Y)}}$$

Amdahl's Law

$$\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \times \left[\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right]$$

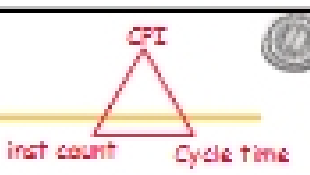
$$\text{Speedup}_{\text{overall}} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Best you could ever hope to do:

$$\text{Speedup}_{\text{maximum}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right)}$$

Today:
Quick review of everything you should have learned

Computer Performance



$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization		X	X
Technology			X

Cycles Per Instruction (Throughput)

"Average Cycles per Instruction"

$$\text{CPI} = (\text{CPU Time} * \text{Clock Rate}) / \text{Instruction Count} = \text{Cycles} / \text{Instruction Count}$$

$$\text{CPU time} = \text{Cycle Time} \times \sum_{j=1}^{n} \text{CPI}_j \times I_j$$

$$\text{CPI} = \sum_{j=1}^{n} \text{CPI}_j \times F_j \quad \text{where } F_j = \frac{I_j}{\text{Instruction Count}}$$

"Instruction Frequency"

Example: Calculating CPI bottom up

Run benchmark and collect workload characterization (simulate, machine counters, or sampling)

Base Machine (Reg / Reg)

Op	Freq	Cycles	CPI(i)	(% Time)
ALU	50%	1	.5	(33%)
Load	20%	2	.4	(27%)
Store	10%	2	.2	(13%)
Branch	20%	2	.4	(27%)
			1.5	

Typical Mix of instruction types in program

Design guideline: Make the common case fast

MIPS 1% rule: only consider adding an instruction of it is shown to add 1% performance improvement on reasonable benchmarks.

Example: Branch Stall Impact

- Assume CPI = 1.0 ignoring branches (ideal)
- Assume solution was stalling for 3 cycles
- If 30% branch, Stall 3 cycles on 30%

Op	Freq	Cycles	CPI(i)	(% Time)
Other	70%	1	.7	(37%)
Branch	30%	4	1.2	(63%)

⇒ new CPI = 1.9

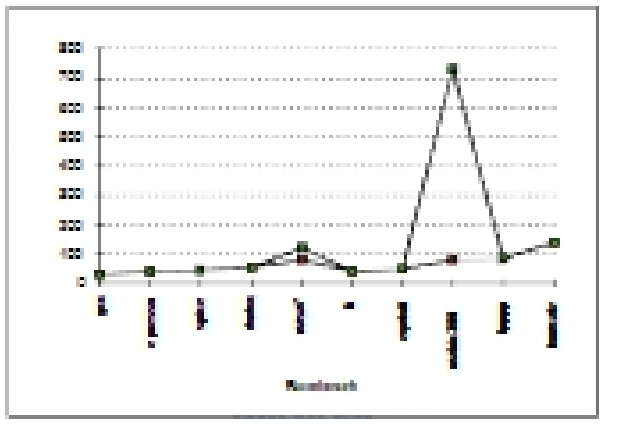
- New machine is 1/1.9 = 0.52 times faster (i.e. slow!)

SPEC: System Performance Evaluation Cooperative

- First Round 1989
  - 10 programs yielding a single number ("SPECmarks")
- Second Round 1992
  - SPECint92 (6 integer programs) and SPECfp92 (14 floating point programs)
    - Compiler Flags unlimited. March 93 of DEC 4000 Model 610:
      spice: unix.c:/def=(sgen,bas_bcopy,"bcopy(a,b,c)=memcpy(b,a,c)"
      wave5: /all=(all,dcom=nat)/ag=a/ur=4/ur=200
      nasa7: /noassu/ag=a/ur=4/ur2=200/lc=blas
- Third Round 1995
  - new set of programs: SPECint95 (8 integer programs) and SPECfp95 (10 floating point)
  - "benchmarks useful for 3 years"
  - Single flag setting for all programs: SPECint_base95, SPECfp_base95
- Fourth Round 2000: 26 apps
  - analysis and simulation programs
  - Compression: bzip2, gzip,
  - Integrated circuit layout, ray tracing, lots of others

SPEC First Round

- One program: 99% of time in single line of code
- New front-end compiler could improve dramatically

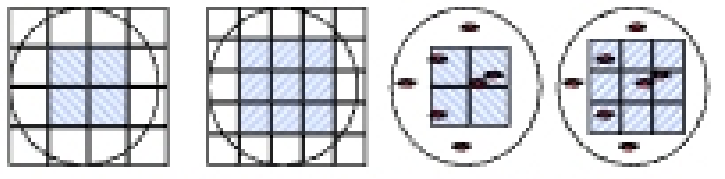


Integrated Circuits Costs

$$\text{IC cost} = \frac{\text{Die cost} + \text{Testing cost} + \text{Packaging cost}}{\text{Final test yield}}$$

$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} \times \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi \times (\text{Wafer\_diam}/2)^2}{\text{Die\_Area}} - \frac{\pi \times \text{Wafer\_diam}}{\sqrt{2} \cdot \text{Die\_Area}} - \text{Test\_die}$$



$$\text{Die Yield} = \text{Wafer\_yield} \times \left\{1 + \left(\frac{\text{Defect\_Density} \times \text{Die\_area}}{\alpha}\right)\right\}^{-\alpha}$$

Die Cost goes roughly with die area^4

A "Typical" RISC

- 32-bit fixed format instruction (3 formats)
- 32 32-bit GPR (R0 contains zero, DP take pair)
- 3-address, reg-reg arithmetic instruction
- Single address mode for load/store: base + displacement
  - no indirection
- Simple branch conditions
- Delayed branch

see: SPARC, MIPS, HP PA-Risc, DEC Alpha, IBM PowerPC, CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3

Example: MIPS (- DLX)

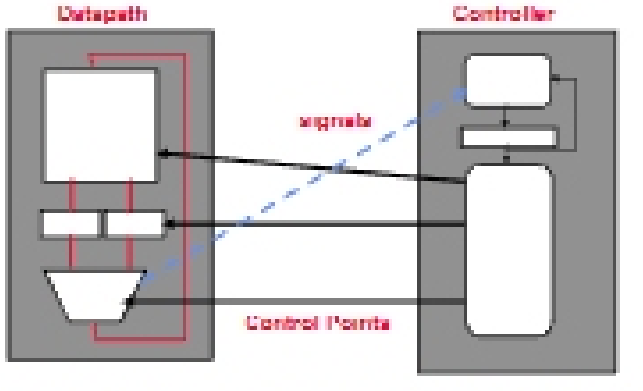
Register-Register

Register-Immediate

Branch

Jump / Call

Datapath vs Control



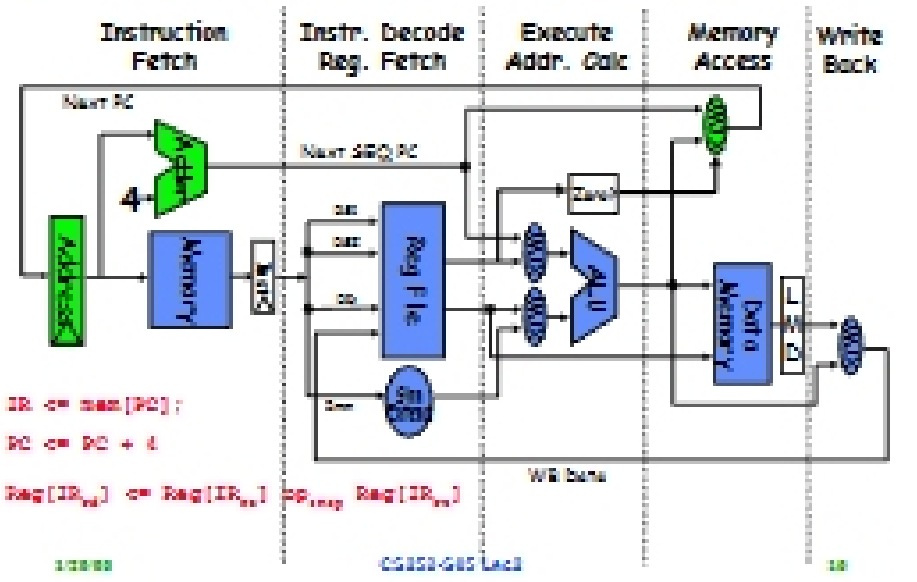
- Datapath: Storage, FU, Interconnect sufficient to perform the desired functions
  - Inputs are Control Points
  - Outputs are signals
- Controller: State machine to orchestrate operation on the data path
  - Based on desired function and signals

Approaching an ISA

- Instruction Set Architecture
  - Defines set of operations, instruction format, hardware supported data types, named storage, addressing modes, sequencing
- Meaning of each instruction is described by RTL on *architected registers* and memory
- Given technology constraints assemble adequate datapath
  - Architected storage mapped to actual storage
  - Function units to do all the required operations
  - Possible additional storage (eg. MAR, MBR, ...)
  - Interconnect to move information among regs and FUs
- Map each instruction to sequence of RTLs
- Collate sequences into symbolic controller state transition diagram (STD)
- Lower symbolic STD to control points
- Implement controller

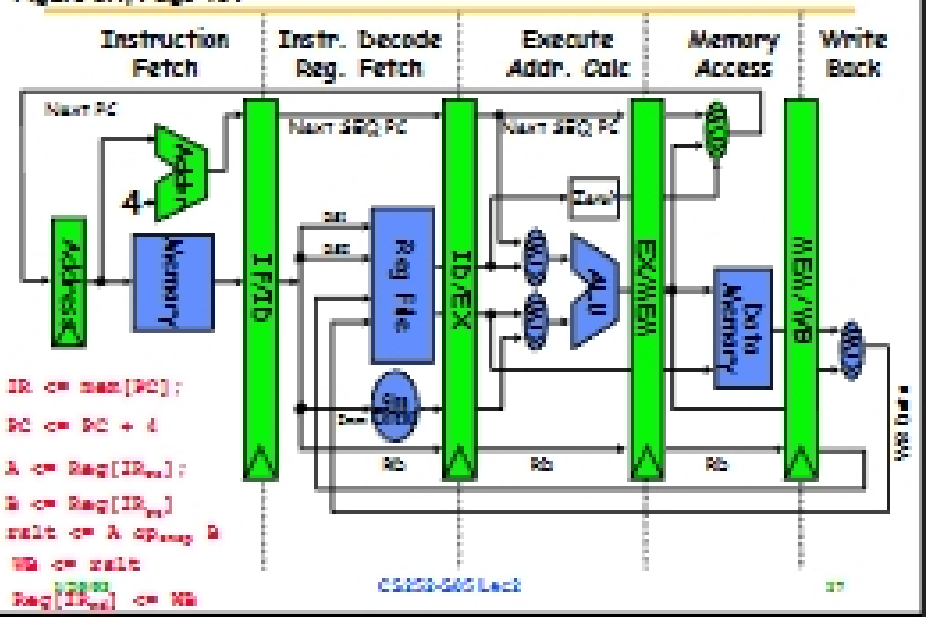
5 Steps of DLX Datapath

Figure 3.1, Page 130



5 Steps of DLX Datapath

Figure 3.4, Page 134



Inst. Set Processor Controller