ESE532: System-on-a-Chip Architecture

Day 14: October 17, 2018

VLIW

(Very Long Instruction Word Processors)

Today

VLIW (Very Large Instruction Word)

Exploiting Instruction-Level Parallelism (ILP)

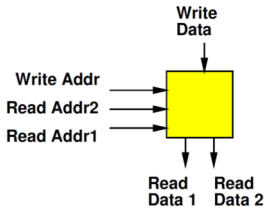
- Demand
- Basic Model
- Costs
- Tuning

Message

- VLIW as a Model for
  - Instruction-Level Parallelism (ILP)
  - Customizing Datapaths
  - Area-Time Tradeoffs

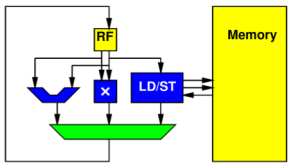
Register File

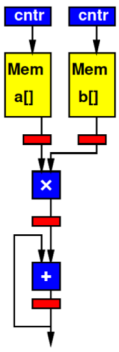
- Small Memory
- Usually with multiple ports
  - Ability to perform multiple reads and writes simultaneously
- Small
  - To make it fast (small memories fast)
  - Multiple ports are expensive



Preclass 1

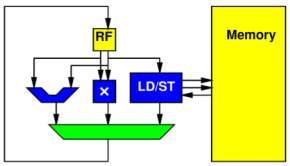
- Cycles per multiply-accumulate
  - Spatial Pipeline
  - Processor

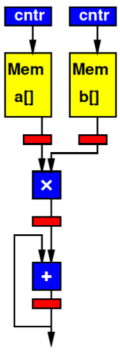




Preclass 1

- How different?
  - Resources
  - Ability to use resources

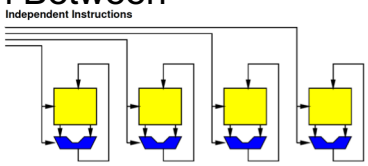




Computing Forms

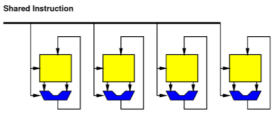
- Processor – does one thing at a time
- Spatial Pipeline – can do many things, but always the same
- Vector – can do the same things on many pieces of data

In Between



What if…

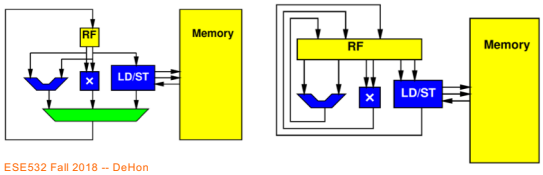
- Want to
  - Do many things at a time (ILP)
  - But not the same (DLP)



In Between

What if…

- Want to
  - Do many things at a time (ILP)
  - But not the same (DLP)
- Want to use resources concurrently



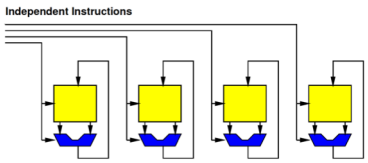
In Between

What if…

- Want to
  - Do many things at a time (ILP)
  - But not the same (DLP)
- Want to use resources concurrently
- Want to
  - Accelerate specific task
  - But not go to spatial pipeline extreme

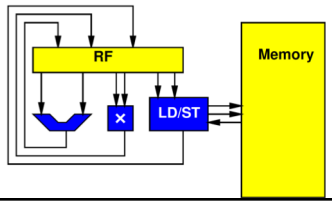
VLIW Feature: Supply Independent Instructions

- Provide instruction per ALU (resource)
- Instructions more expensive than Vector
  - But more flexible



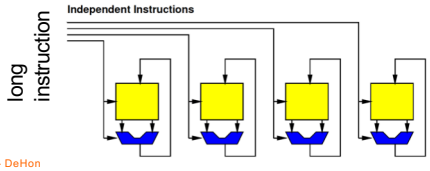
Control Heterogeneous Units

- Control each unit simultaneously and independently
  - More expensive than processor
    - Memory ports and/or interconnect
  - But more parallelism



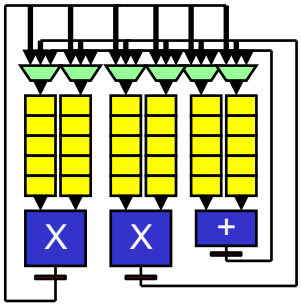
VLIW

- The "instruction"
  - The bits controlling the datapath
- …becomes long
- Hence:
  - Very Long Instruction Word (VLIW)



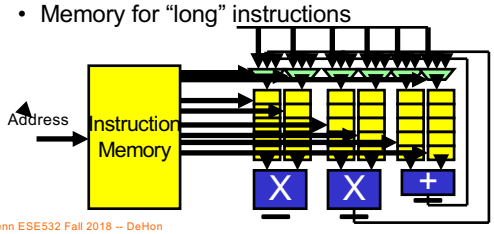
VLIW

- Very Long Instruction Word
- Set of operators
  - Parameterize number, distribution (X, +, sqrt…)
    - More operators→ less time, more area
    - Fewer operators→ more time, less area
- Memories for intermediate state

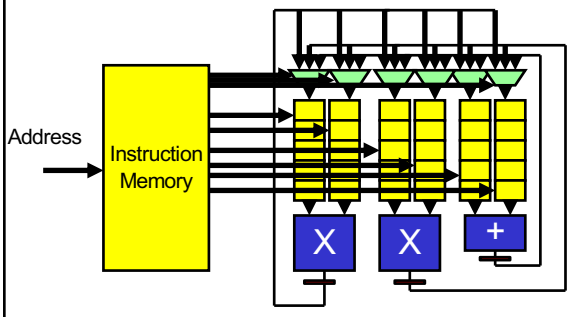


VLIW

- Very Long Instruction Word
- Set of operators
  - Parameterize number, distribution (X, +, sqrt…)
    - More operators→ less time, more area
    - Fewer operators→ more time, less area
- Memories for intermediate state
- Memory for "long" instructions



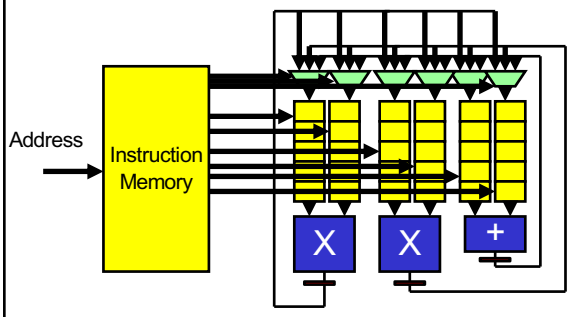
VLIW



VLIW

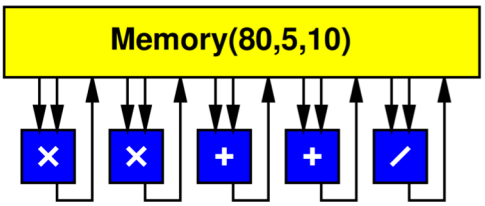
- Very Long Instruction Word
- Set of operators
  - Parameterize number, distribution (X, +, sqrt…)
    - More operators→ less time, more area
    - Fewer operators→ more time, less area
- Memories for intermediate state
- Memory for "long" instructions
- General framework for specializing to problem
  - Wiring, memories get expensive
  - Opportunity for further optimizations
- General way to tradeoff area and time

VLIW



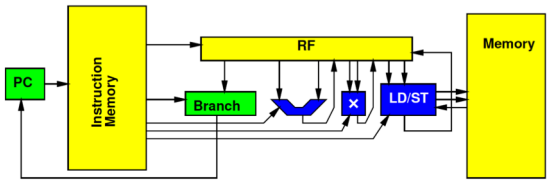
VLIW w/ Multiport RF

- Simple, full-featured model use common Register File
  - Memory(Words, WritePorts, ReadPorts)



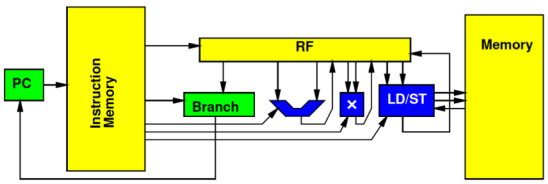
Processor Unbound

- Can (design to) use all operators at once



Processor Unbound

- Implement Preclass 1

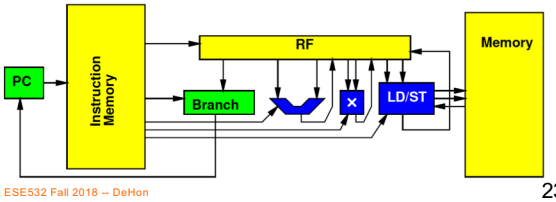


Schedule

Cycle	Branch	ALU	Multiply	LD/ST
0	Bzneq r3,end	Add r4		
1		Add r5		Ld r4,r6
2		Sub r2,r1,r3		Ld r5,r7
3		Add r1,#1,r1	Mpy r7,r8,r8	
4	B top	Add r7,r8,r8		

VLIW Operator Knobs

- Choose collection of operators and the numbers of each
  - Match task
  - Tune resources



Schedule

- Choose collection of operators and the numbers of each
  - Match task
  - Tune resources

What operator might we add to accelerate this loop?

Cycle	Branch	ALU	Multiply	LD/ST
0	Bzneq r3,end	Add r4		
1		Add r5		Ld r4,r6
2		Sub r2,r1,r3		Ld r5,r7
3		Add r1,#1,r1	Mpy r7,r8,r8	
4	B top	Add r7,r8,r8		

Preclass 2a

- res[i]=sqrt(x[i]*x[i]+y[i]*y[i]+z[i]*z[i]);
- II with one operator of each?

Schedule

Cycle	LD	ST	Multiply	Add	incr	sqrt
0				i<MAX	&X[i]	
1	X[i]				&Y[i]	
2	Y[i]		X[i]*X[i]		&Z[i]	
3	Z[i]		Y[i]*Y[i]			
4			Z[i]*Z[i]	X^2+Y^2		
5				$(X^2+Y^2)+Z^2$		
6						Sqrt()
7		Res[i]			i	

Preclass 2b

- res[i]=sqrt(x[i]*x[i]+y[i]*y[i]+z[i]*z[i]);
- Minimum II achievable?
  - Latency lower bound

Critical Path

- Increment pointers / branch
- Load
- Multiplies
- Add
- Add
- Squareroot
- Writeback

Preclass 2c

- res[i]=sqrt(x[i]*x[i]+y[i]*y[i]+z[i]*z[i]);
- How many operators of each type to achieve minimum II (latency lower bound)?

Schedule w/ 2d Resources

	LD	LD	LD	ST	*	*	*	+	i	i	i	sqrt
0								<	&x	&y	&z	
1	X[i]	Y[i]	Z[i]									
2					x	y	z					
3								X+y				
4								+z				
5												sqrt
6				Res [i]					i			

- What is disappointing about this schedule?

Preclass 2d

- res[i]=sqrt(x[i]*x[i]+y[i]*y[i]+z[i]*z[i]);
- res[i+1]=sqrt(x[i+1]*x[i+1]+y[i+1]*y[i+1]+z[i+1]*z[i+1]);
- res[i+2]=sqrt(x[i+2]*x[i+2]+y[i+2]*y[i+2]+z[i+2]*z[i+2]);
- res[i+3]=sqrt(x[i+3]*x[i+3]+y[i+3]*y[i+3]+z[i+3]*z[i+3]);

- Schedule

Unroll 4

	LD	LD	LD	ST	*	*	*	+	+	i	i	i	sqrt
0								<		x0	y0	z0	
1	x0	y0	z0							x1	y1	z1	
2	x1	y1	z1		x0	y0	z0			x2	y2	z2	
3	x2	y2	z2		x1	y1	z1	xy0		x3	y2	z3	
4	x3	y2	z3		x2	y2	z2	xy1	+z0				
5					x3	y2	z3	xy2	+z1				0
6				0				xy3	+z2				1
7				1					+z3				2
8				2									3
9				3						i			

Time Points

- 4 iterations in 10 cycles = 2.5 cycles/iter
- Compared to 1 iteration in 7
- Compared to 1 iteration in 8

Preclass 2e

- res[i]=sqrt(x[i]*x[i]+y[i]*y[i]+z[i]*z[i]);

- Area comparison?

Midterm

Midterm

- Analysis
  - Bottleneck
  - Amdhal's Law Speedup
  - Computational requirements
  - Resource Bounds
  - Critical Path
  - Latency/throughput/II
- Will be calculating/estimating runtimes
- From Code
- Forms of Parallelism
- Dataflow, SIMD, hardware pipeline, threads
- Pipelining/Retiming
- Map/schedule task graph to (multiple) target substrates
- Memory assignment and movement
- Area-time points

Midterm

- Closed book, notes, etc.
- Calculators allowed (encouraged)
- Last two midterms, final online
  - Both without answers (for practice)
  - …and with answers (check yourself)
- No VLIW on midterm
  - But memory fair game; II, latency…

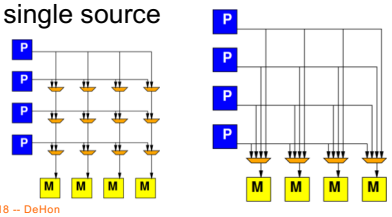
Data Storage and Movement

Multiport RF

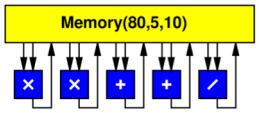
- Multiported memories are expensive
  - Need input/output lines for each port
  - Makes large, slow
- Simplified preclass model:
  - Area(Memory(n,w,r))=n*(w+r+1)/2

Alternate: Crossbar

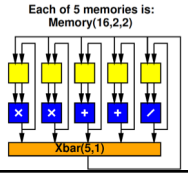
- Provide programmable connection between all sources and destinations
- Any destination can be connected to any single source



Preclass 3



- Operator area?
- Xbar(5,1) area
- Memory area, each case
- Total area
- How does area of memories, xbar compare to datapath operators in each case?

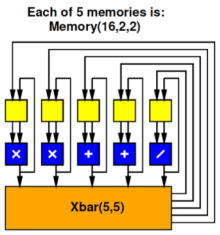


Split RF Cheaper

- At same capacity, split register file cheaper
  - 2R+1W → 2 per word
  - 5R+10W → 8 per word

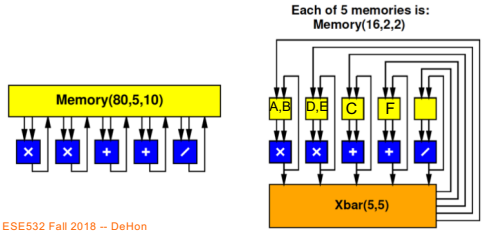
Split RF

- Xbar(5,5) cost?
- Total Area?



Split RF Full Crossbar

- Cycles each for: (A*B+C)/(D*E+F)
  - Assume A..F start as shown

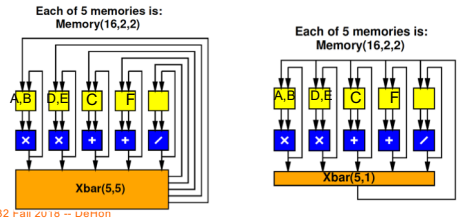


VLIW Memory Tuning

- Can select how much sharing or independence in local memories

Split RF, Limited Crossbar

- What limitation does the one crossbar output pose?
  - Cycles for same task: (A*B+C)/(D*E+F)



VLIW Schedule

Need to schedule Xbar output(s) as well as operators.

cycle	*	*	+	+	/	Xbar
0						
1						
2						
3						
4						

VLIW vs. Superscalar

VLIW vs. SuperScalar

- Modern, high-end proc. (incl. ARM on Zynq)
  - Do support ILP
  - Issue multiple instructions per cycle
  - …but, from a single, sequential instruction stream
- SuperScalar – dynamic issue and interlock on data hazards – hide # operators
  - Must have shared, multiport RF
- VLIW – offline scheduled
  - No interlocks, allow distributed RF
  - Lower area/operator – need to recompile code

Back to VLIW

Pipelined Operators

- Often seen, will have pipelined operators
  - E.g. 3 cycles multiply
- How complicate?

Accommodating Pipeline

- Schedule for when data becomes available
  - Dependencies
  - Use of resources

cycle	*	*	+	+	/	Xbar
0	X*X					
1	Y*Y					
2						X*X
3						Y*Y
4			X^2+Y^2			X^2+Y^2
5					X^2+Y^2/Z	
6						

Accommodating Pipeline

- Schedule for when data becomes available
  - Dependencies
  - Use of resources

Impossible schedule; Conflict on single Xbar output

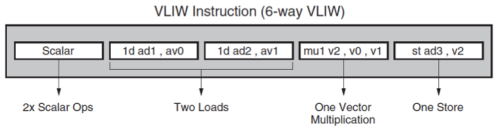
cycle	*	*	+	+	/	Xbar
0	X*X					
1	Y*Y					
2						X*X
3			Q+R			Y*Y,Q+R
4			X^2+Y^2			X^2+Y^2
5					X^2+Y^2/Z	
6						

VLIW Interconnect Tuning

- Can decide how rich to make the interconnect
  - Number of outputs to support
  - How to depopulate crossbar
  - Use more restricted network

Commercial: Xilinx AI Engine

- 6-way superscalar Vector



https://www.xilinx.com/support/documentation/white_papers/wp506-ai-engine.pdf

Xilinx WP506

Big Ideas:

- VLIW as a Model for
  - Instruction-Level Parallelism (ILP)
  - Customizing Datapaths
  - Area-Time Tradeoffs
- Customize VLIW
  - Operator selection
  - Memory/register file setup
  - Inter-functional unit communication network

Admin

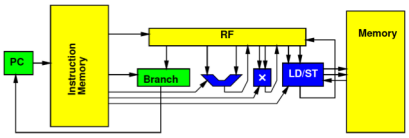
- Midterm on Monday
  - Previous midterms and solutions online
- Extra Review Office Hours on Sunday
  - See Piazza
- HW6 due Friday
  - Remember many slow builds
- HW7 out

Loop Overhead

Bonus slides:
not expect to cover in lecture

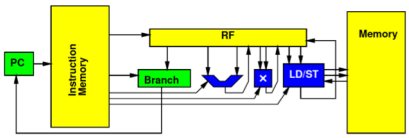
Loop Overhead

- Can handle loop overhead in ILP on VLIW
  - Increment counters, branches as independent functional units



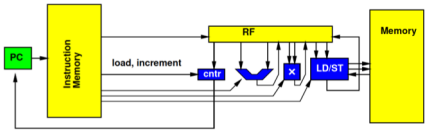
VLIW Loop Overhead

- Can handle loop overhead in ILP on VLIW
- …but paying a full issue unit and instruction costs overhead

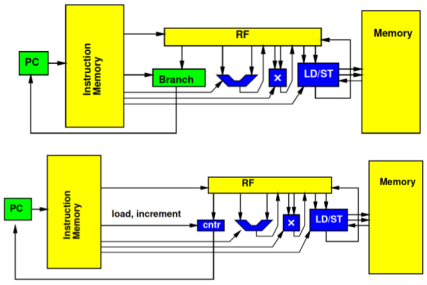


Zero-Overhead Loops

- Specialize the instructions, state, branching for loops
  - Counter rather than RF
  - One bit to indicate if counter decrement
  - Exit loop when decrement to 0

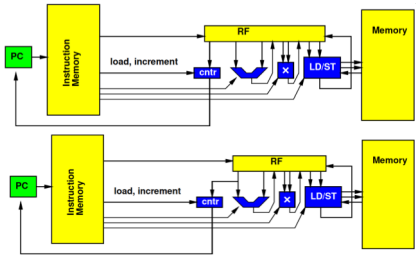


Simplification



Zero-Overhead Loop Simplify

- Share port – simplify further



Zero-Overhead Loop Example (preclass 1)

repeat r3:

```
addi r4,#4,r4;
addi r5,#4,r5; ld r4,r6
ld r5,r7
mul r6,r7,r7
add  r7,r8,r8
```


Zero-Overhead Loop

- Potentially generalize to multiple loop nests and counters
- Common in highly optimized DSPs, Vector units