ESE532: System-on-a-Chip Architecture

Day 13: October 16, 2017
VLIW
(Very Long Instruction Word Processors)

Today

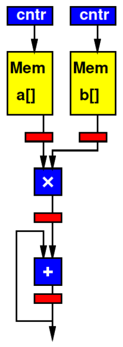
VLIW (Very Large Instruction Word)
- Demand
- Basic Model
- Costs
- Tuning

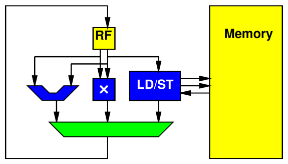
Message

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

Preclass 1

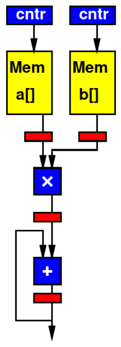
- Cycles per multiply-accumulate
  - Spatial Pipeline
  - Processor

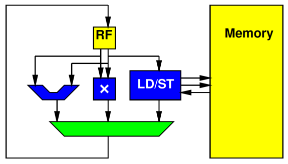




Preclass 1

- How different?





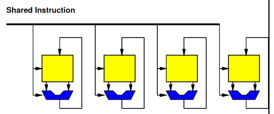
Computing Forms

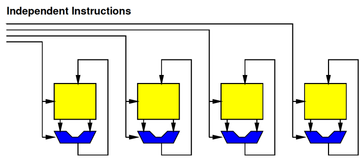
- Processor – does one thing at a time
- Spatial Pipeline – can do many things, but always the same
- Vector – can do the same things on many pieces of data

In Between

What if…

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

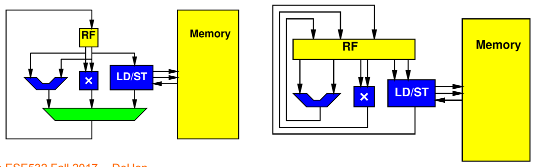




In between

What if…

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



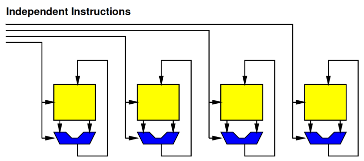
In between

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

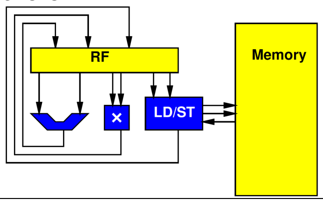
Supply Independent Instructions

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



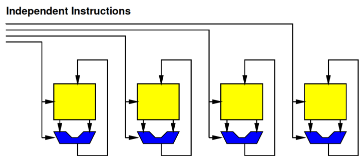
Control Heterogeneous Units

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



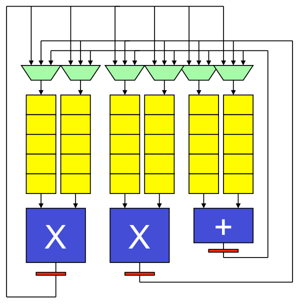
VLIW

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



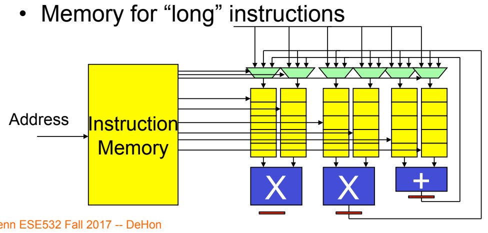
VLIW

- 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

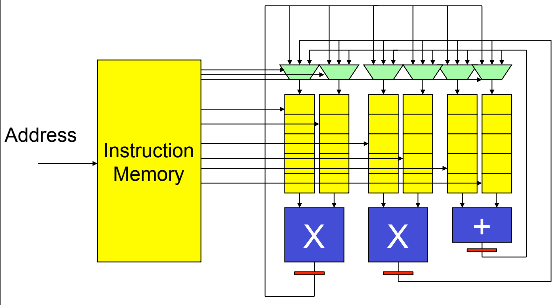


VLIW

- 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



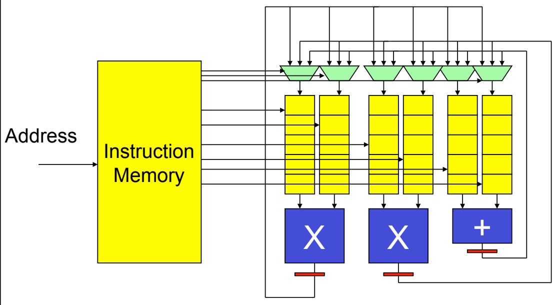
VLIW



VLIW

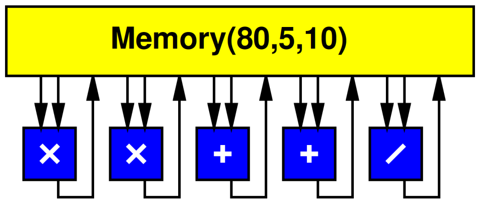
- Very Long Instruction Word
- Set of operators
  - Parameterize number, distribution (X, +, sqrt…)
    - More operators→ less time, more area
    - Fewer operators→ more time, less area
- Memories for intermediate state
- Memory for "long" instructions
- General framework for specializing to problem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

VLIW



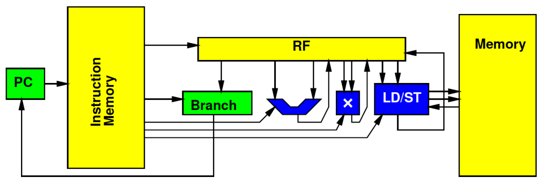
VLIW w/ Multiport RF

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



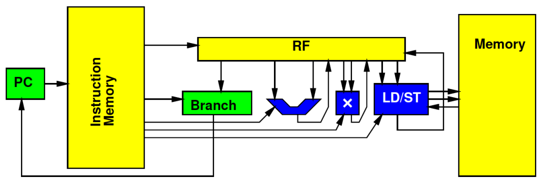
Processor Unbound

- Can (design to) use all operators at once



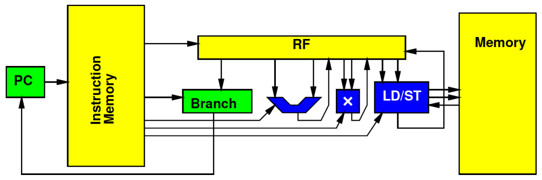
Processor Unbound

- Implement Preclass 1



VLIW Operator Knobs

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



Preclass 2

- res[i]=sqrt(x[i]*x[i]+y[i]*y[i]+z[i]*z[i]);
- II with one operator of each?
- Minimum II achievable?
  - Latency lower bound
- How many operators of each type?
- Area comparison?

Critical Path

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

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]);

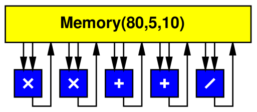
Time Points

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

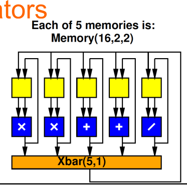
Multiport RF

- 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

Preclass 3



- Compare total area
  - Multiport 5, 10
  - 5 x Multiport 2, 2 with 5x1 Xbar
- How does area of memories, xbar compare to datapath operators in each case?

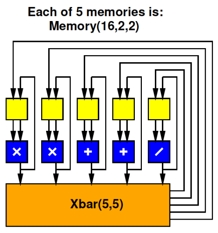


Split RF Cheaper

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

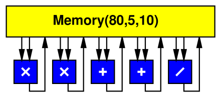
Split RF

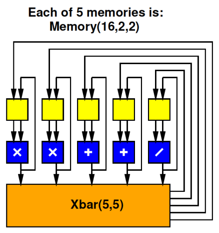
- Split RF with Full (5, 5) Crossbar
  - Cost?



Split RF Full Crossbar

- What restriction/limitation might this have versus multiported RF version?



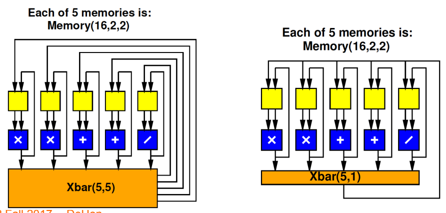


VLIW Memory Tuning

- Can select how much sharing or independence in local memories

Split RF, Limited Crossbar

- What limitation does the one crossbar output pose?


Each of 5 memories is: Memory(16,2,2)
Xbar(5,5)
Each of 5 memories is: Memory(16,2,2)
Xbar(5,1)

VLIW Schedule

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

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

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

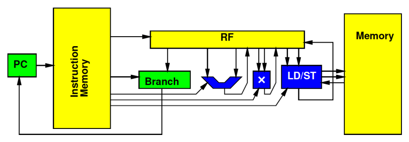
cycle	*	*	+	+	/	Xbar
0	X*X					
1	Y*Y					
2						X*X
3			Q+R			Y*Y,Q +R
4			X^2+Y^2			X^2+Y^2
5					$X^2+Y^2/$ Z	

VLIW Interconnect Tuning

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

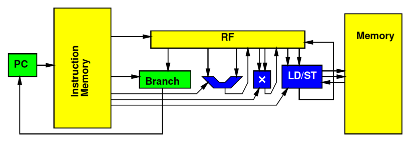
Loop Overhead

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



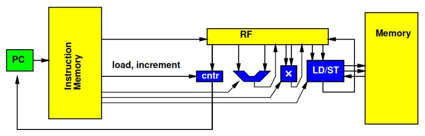
VLIW Loop Overhead

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

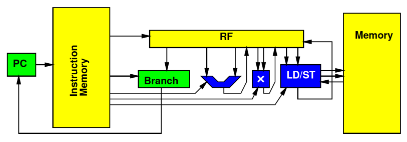


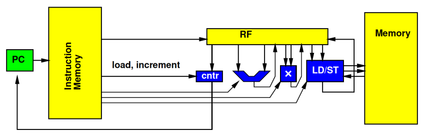
Zero-Overhead Loops

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



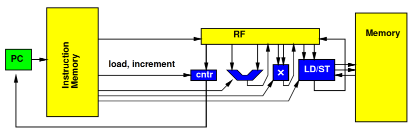
Simplification

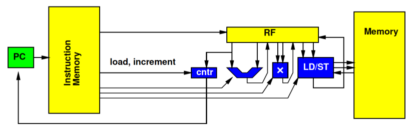




Zero-Overhead Loop Simplify

- 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

VLIW vs. SuperScalar

- Modern, high-end processors
  - 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

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

- Reading for Wed. online
- HW6 due Friday
  - Remember many slow builds
- Midterm next Monday
  - See Spring 2017 syllabus for
    - Last semesters midterm and final
      - …with solutions