ESE532: System-on-a-Chip Architecture

Day 15: March 15, 2017
VLIW
(Very Long Instruction Word Processors)

Today

VLIW (Very Large Instruction Word)

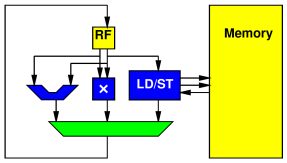
- Demand
- Basic Model
- Costs
- Tuning

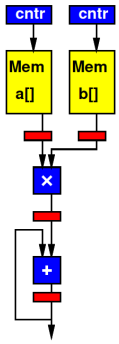
Message

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

Preclass 1

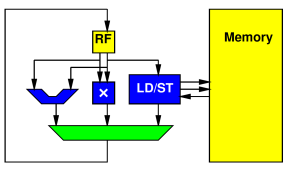
- Cycles per multiply-accumulate
  - Spatial Pipeline
  - Processor

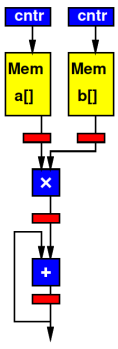




Preclass 1

- How different?

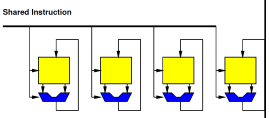




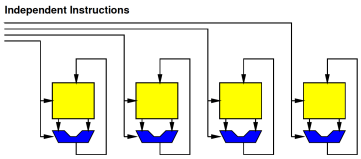
Computing Forms

- 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

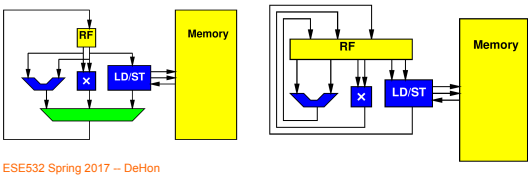


What if…
- Want to
  - Do many things at a time (ILP)
  - But not the same (DLP)



In between

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

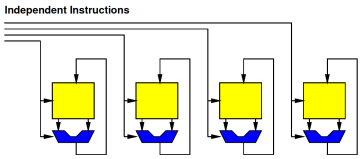


In between

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

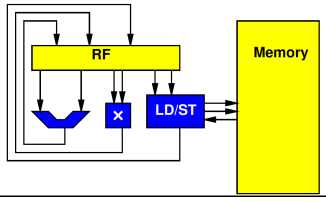
Supply Independent Instructions

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



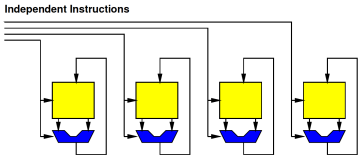
Control Heterogeneous Units

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



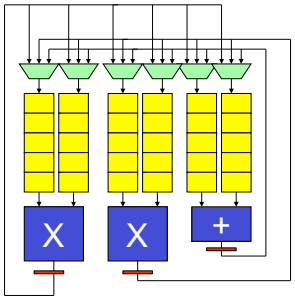
VLIW

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



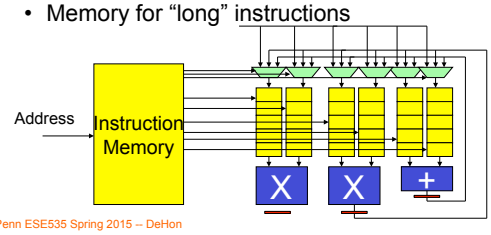
VLIW

- 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

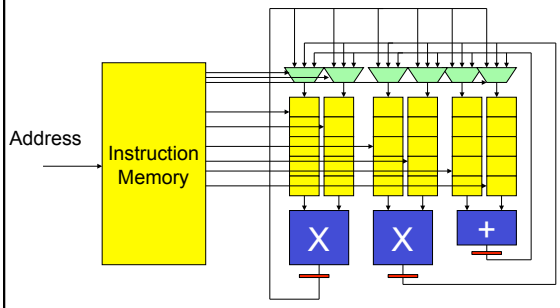


VLIW

- 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



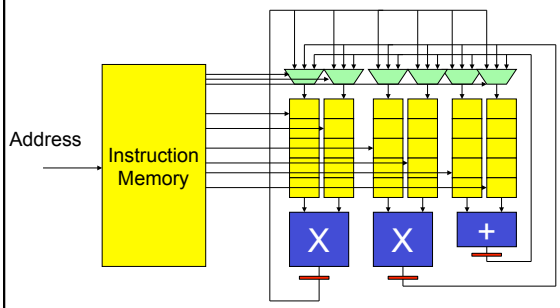
VLIW



VLIW

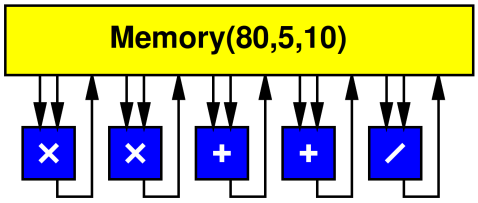
- Very Long Instruction Word
- Set of operators
  - Parameterize number, distribution (X, +, sqrt…)
    - More operators→ less time, more area
    - Fewer operators→ more time, less area
- Memories for intermediate state
- Memory for "long" instructions
- General framework for specializing to problem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

VLIW



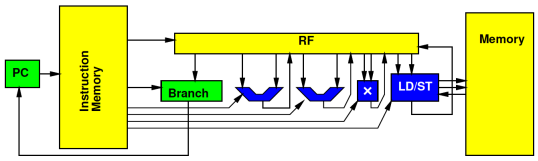
VLIW w/ Multiport RF

- Simple, full-featured model use common Register File



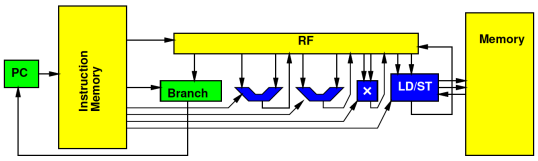
Processor Unbound

- Can (design to) use all operators at once



Processor Unbound

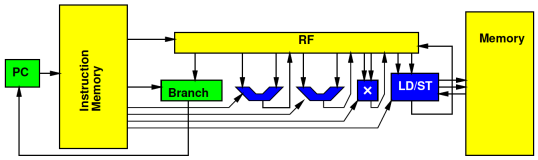
- Implement Preclass 1



Software Pipelined Version

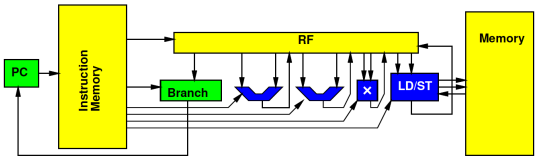
```
for (i=0;i<MAX;i++)
  { c=c+prod; prod=la*lb; la=a[i]; lb=b[i];}
```

- Use this to compact schedule



VLIW Operator Knobs

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



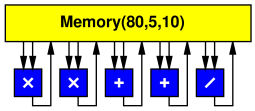
Preclass 2

- res[i]=sqrt(x[i]*x[i]+y[i]*y[i]+z[i]*z[i]);
- How many operators need for each II? Datapath Area?

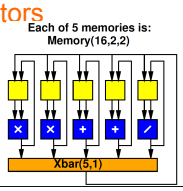
Multiport RF

- 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?

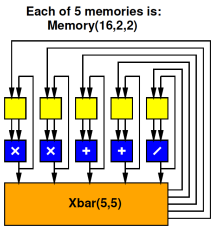


Split RF Cheaper

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

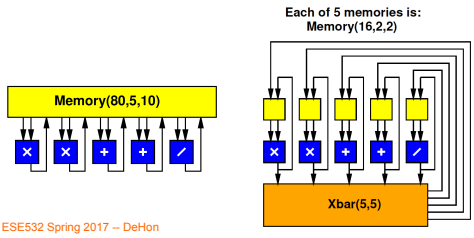
Split RF

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



Split RF Full Crossbar

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

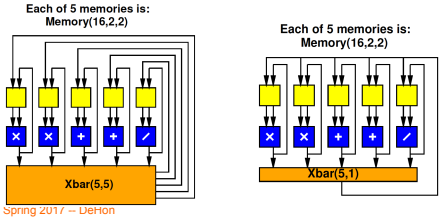


VLIW Memory Tuning

- Can select how much sharing or independence in local memories

Split RF, Limited Crossbar

- What limitation does the one crossbar output pose?



VLIW Schedule

Need to schedule Xbar output(s) as well as operators.
(as seen Day 12)

Cycle	0	1	2	3	4	5	6	7	8	9
X 1										
X 2										
+ 1										
+ 2										
/										
Xbar										

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

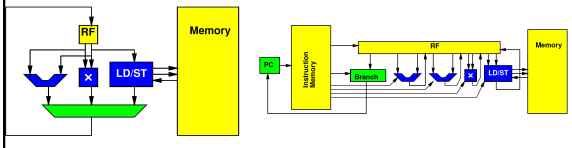
Cycle	0	1	2	3	4	5	6	7	8	9
X 1										
X 2	OP									
+ 1			AOP							
+ 2										
/										
Xbar			AOP	OP						

VLIW Interconnect Tuning

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

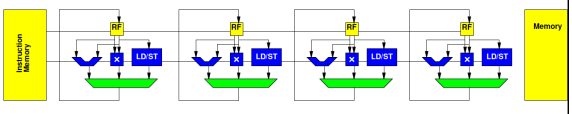
Compare

- Compare processor and unbound-VLIW processor under preclass 3 model.
  - 32 registers, Mux 3x1 crossbar
  - Branch=10, LD/ST=200, ALU=10
  - Assume Imem, Memcomparable



Compare

- Vector and VLIW under preclass 3 model
  - LD/ST=200 (only charge 1 across vector)
  - Charge per multiplier/adder/Multiport 16,1,2 RF
  - (hint: 1 same as proc, 3 same – ld/st)



Compare Vector VLIW

- Instructions for dot product (preclass 1)
  - 10 instructions for 4 multiply-adds (VL=4)

```
top: sub r1,r2,r3 // r3=MAX-i
     bzneq r3, end
     addi r4,#16,r4
     addi r5,#16,r5
     vld r4,v6 // a[i]…a[i+3]
     vld 56,v7 // b[i]…b[i+3]
     vmul v6,v7,v7 // a[i]*b[i]
     vadd v7,v8,v8 // c+=a[i]*b[i]
     addi r1,#4,r1 // i+=4
end: vreduce v8,r8
```

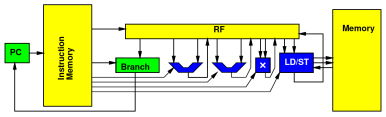

Loop Overhead

- Looping, incrementing pointers costs a few instructions
- Can be large overhead in tight loops
- Most/all of non-vector operations in previous example

```
top: sub r1,r2,r3 // r
     bzneq r3, end //
     addi r4,#4,r4 //&
     addi r5,#4,r5 //&
     ld r4,r6 // a[i]
     ld r5,r7 // b[i]
     mul r6,r7,r7 // a
     add r7,r8,r8 // c
     addi r1,#1,r1 //
end:
```

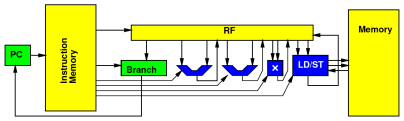

Loop Overhead

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



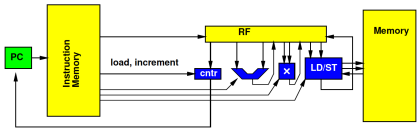
VLIW Loop Overhead

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

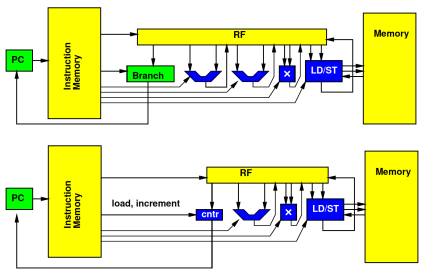


Zero-Overhead Loops

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

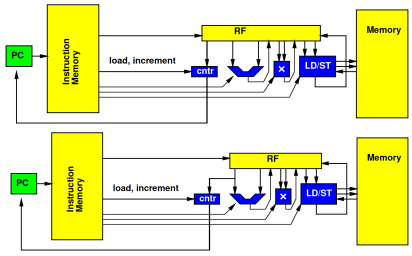


Simplification



Zero-Overhead Loop Simplify

- Share port – simplify further



Zero-Overhead Loop Example (preclass 1)

repeat r3:

addi r5,#4,r5; ld r4,r5

addi r4,#4,r4; ld r6,r7

add r9,r8,r8; mul r6,r7,r9

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
  - 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

- HW7 due Friday
  - Individual