# ESE532: System-on-a-Chip Architecture

Day 26: December 7, 2020

VLIW

(Very Long Instruction Word Processors)



## Today

VLIW (Very Large Instruction Word)

Exploiting Instruction-Level Parallelism (ILP)

- Part 1:
  - Demand, Basic Model
- Part 2:
  - Tuning, Unrollling
- Part 3:
  - Data Movement
- Part 4 (bonus, probably not cover):
  - Zero-Overhead Loops, Pipelined Operators



## Message

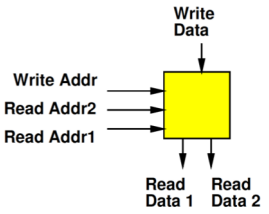
- VLIW as a Model for
  - Instruction-Level Parallelism (ILP)
  - Customizing Datapaths
  - Area-Time Tradeoffs





## Register File

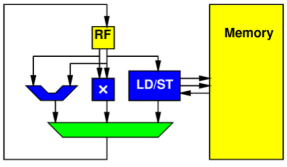
- Small Memory
- Usually with multiple ports
  - Ability to perform multiple reads and writes simultaneously
- Small
  - To make it fast (small memories fast)
  - Multiple ports are expensive

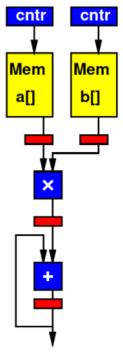





## Preclass 1

- Cycles per multiply-accumulate
  - Spatial Pipeline
  - Processor

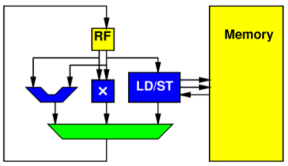


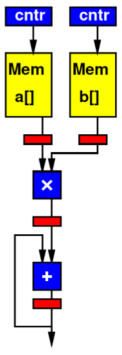






## Preclass 1

- How different?
  - Resources
    - Number of units can perform adds
    - Number of simultaneous memory reads
  - Ability to use resources
    - What resources can use in same cycle



## 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

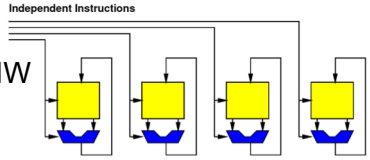


## In Between

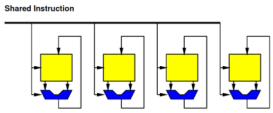
What if…

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

VLIW




SIMD

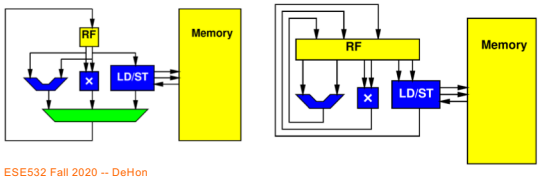





## In Between

What if…

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






## In Between

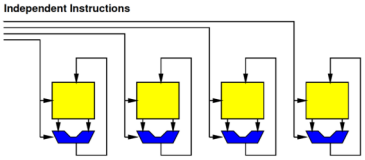
What if…

- 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

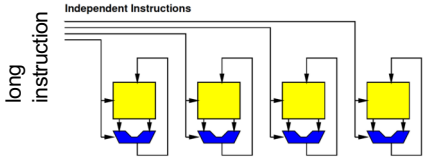
- Provide instruction per ALU (resource)
- Instructions more expensive than Vector
  - But more flexible





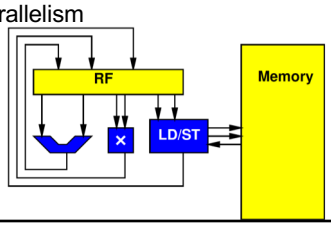

## VLIW

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



## Control Heterogeneous Units

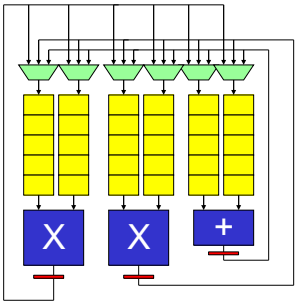
- Control each unit simultaneously and independently
  - More expensive than processor
    - Memory ports and/or interconnect
  - But more parallelism






## VLIW

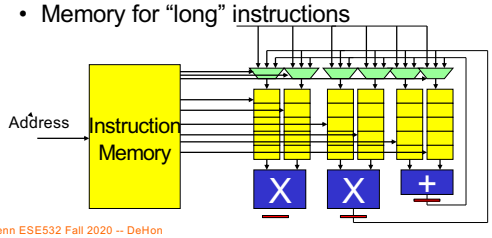
- Very Long Instruction Word
- Set of operators
  - Parameterize number, distribution (X, +, sqrt…)
    - More operators→ less time, more area
    - Fewer operators→ more time, less area
- Memories for intermediate state





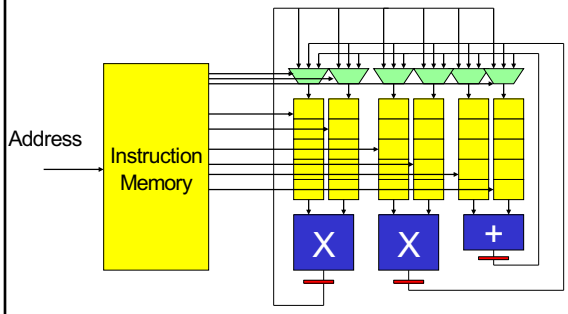

## VLIW

- 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



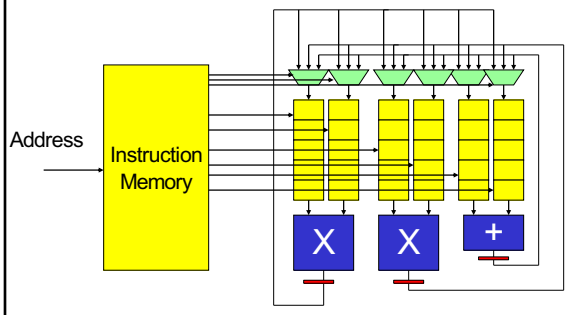



## VLIW

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

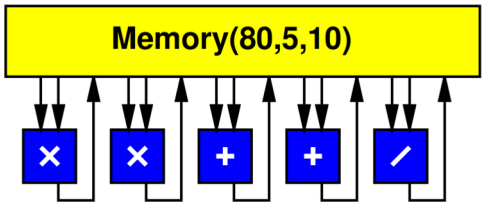


## VLIW



## VLIW w/ Multiport RF

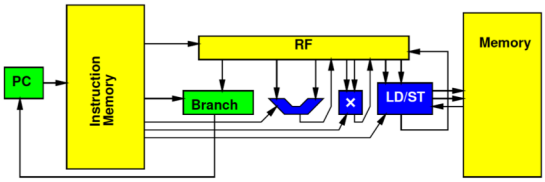
- Simple, full-featured model use common Register File
  - Memory(Words, WritePorts, ReadPorts)






## Processor Unbound

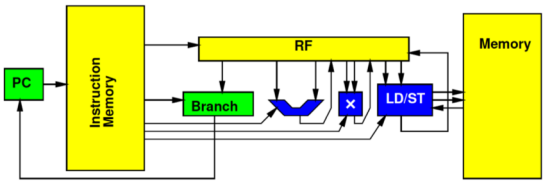
- Can (design to) use all operators at once






## Processor Unbound

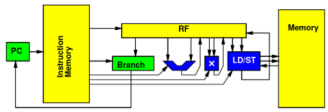
- Schedule Preclass 1 on this VLIW (next slide)






## Schedule Preclass 1

| Cycle | Branch | ALU | Multiply | LD/ST |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |





## Schedule

| Cycle | Branch | ALU | Multiply | LD/ST |
|---|---|---|---|---|
| 0 | | Sub r2,r1,r3 | | |
| 1 | Bzneq r3,end | Add r4 | | |
| 2 | | Add r5 | | Ld r4,r6 |
| 3 | | | | Ld r5,r7 |
| 4 | | Add r1,#1,r1 | Mpy r7,r8,r8 | |
| 5 | B top | Add r7,r8,r8 | | |

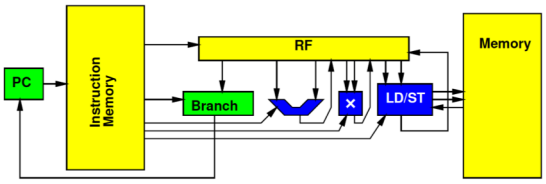


## Tuning, Unrolling

Part 2

# VLIW Operator Knobs (Design Space Axes)

- 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 | | Sub r2,r1,r3 | | |
| 1 | Bzneq r3,end | Add r4 | | |
| 2 | | Add r5 | | Ld r4,r6 |
| 3 | | | | Ld r5,r7 |
| 4 | | Add r1,#1,r1 | Mpy r7,r8,r8 | |
| 5 | 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/ 2c Resources

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



## 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 (next slide)



## Unroll 4 Schedule

| | LD | LD | LD | ST | * | * | * | + | + | i | i | i | sqrt |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | | |



## 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?

| Op | read | write | * | + | incr | sqrt | Area | II |
|---|---|---|---|---|---|---|---|---|
| A | 20 | 30 | 10 | 2 | 1 | 25 | | |
| 2a | 1 | 1 | 1 | 1 | 1 | 1 | | 7 |
| 2c | | | | | | | | 8 |
| 2d | 3 | 1 | 3 | 2 | 3 | 1 | | 2.5 |



## Take Aways

- VLIW: Tune Operators to select Area-Time point
- Running single iteration of loop to completion before start next
  - Inefficient
  - Latency bound for loop
  - Benefit schedule multiple loop bodies



## Data Storage and Movement

Part 3



## Multiport RF

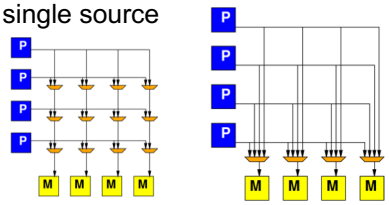
- Multiported memories are expensive
  - Need input/output lines for each port
  - Makes large, slow
- Simplified preclass model:
  - Area(Memory(n,w,r))=n*(w+r+1)/2



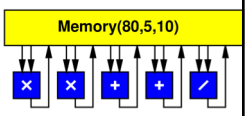


## Alternate: Crossbar (Xbar)

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

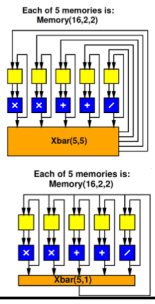


## Preclass 3




- Areas? (table)
- How does area of memories, xbar compare to datapath operators in each case?

| | Multiport RF | Split RF Xbar(5,5) | Split RF Xbar(5,1) |
|---|---|---|---|
| Memory | | | |
| Operators | | | |
| Xbar | | | |
| Total | | | |





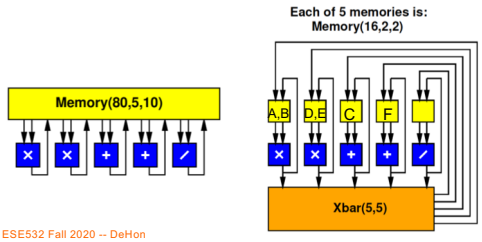

## Split RF Cheaper

- At same capacity, split register file cheaper
  - 2W+2R → 2.5 per word
  - 5W+10R → 8 per word



## Split RF Full Crossbar

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





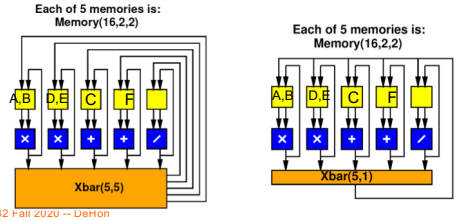

## VLIW Memory Tuning

- 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 | | | | | | |

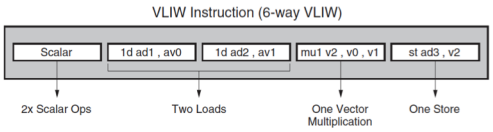
## VLIW Interconnect Tuning

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



## Commercial: Xilinx AI Engine

- 6-way VLIW Vector




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

Xilinx WP506



## 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 (like done today)
  - No interlocks, allow split 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

- Feedback
- Reading for Wednesday online
- Three lectures this week
  - (actual) Thursday is a (logical) Monday
- Project final reports due Thursday
- Final on Friday, Dec. 18th
  - Week from Friday



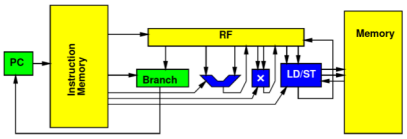
## Loop Overhead

Bonus slides:

not expect to cover in lecture

## Loop Overhead

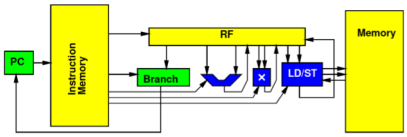
- Can handle loop overhead in ILP on VLIW
  - Increment counters, branches as independent functional units






## VLIW Loop Overhead

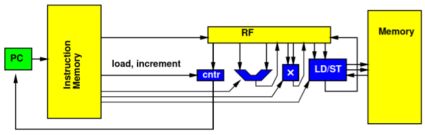
- Can handle loop overhead in ILP on VLIW
- …but paying a full issue unit and instruction costs overhead





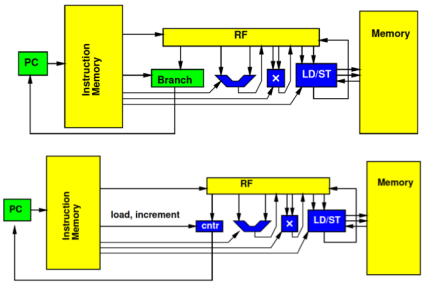

## Zero-Overhead Loops

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





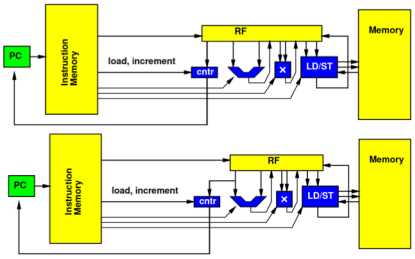

## Simplification






## 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
- Can think about design space options within this VLIW model framework



## Pipelined VLIW Operators



## 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 | | | | | | |



## Pipelined Operators

- Instance of what we saw last time
- Need to schedule use on all resources
- Including
  - Memory port
  - Communications (busses, crossbars ports)
  - I/O