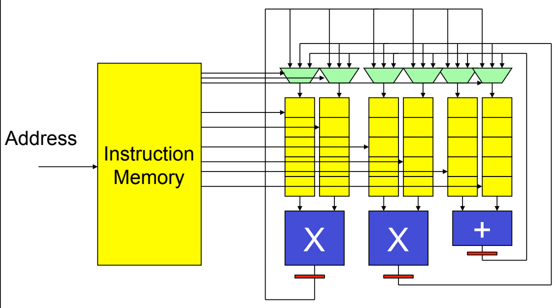
ESE532: System-on-a-Chip Architecture

Day 14: October 18, 2017
Software Pipelining

Previously: 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



Today

- Software Pipelining
- (Midterm topics)
- VLIW (as time permits)
  - Zero-Overhead Loops
  - Conditionals

Message

- Pack computations more tightly by scheduling multiple loop instances together
  - Exploiting pipelining of computation

Problem

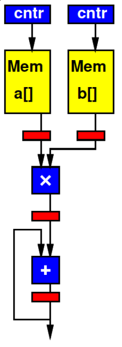
- Low utilization of parallel functional units for a single loop body

	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			

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			

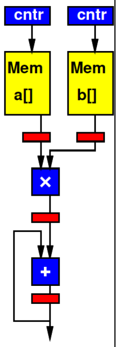
Observation: Pipeline



for (i=0;i<MAX;i++)
    c=+a[i]*b[i];

- When we pipeline we use all the resources
- But, we don't operate on a single loop body instance at a time
- We cannot hit II=1 for VLIW schedule of a single loop body because of path latency

Observation: Pipeline



for (i=0;i<MAX;i++)
    c=+a[i]*b[i];

- Pipeline not operate on a single loop body instance at a time
- Assume add is working on a[0]*b[0] in same cycle,
  - What i is a[i]*b[i]?
  - What i is lookup ld a[i], b[i]?

Observation: Unroll

- Works like a pipeline
- Only works because overlap data among loop instances

	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			

Idea: Software Pipelining

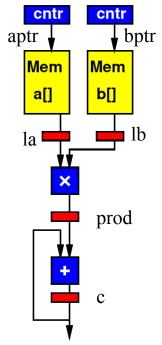
- Schedule VLIW operators across multiple loop iterations
- Treat execution as pipeline

Pipeline

- Fully pipelined, what do in a pipeline stage in a cycle
  - Depends on what did in a different stage on previous cycle
- What do in cycle is a set of pipeline stages
  - Each operating on a different set of input data items

Observation: Pipeline

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



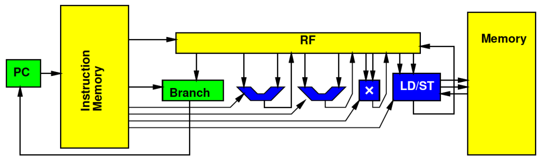
Rewrite body to match cycle of pipeline

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



Schedule Software Pipelined (Preclass 3)

Cycle	Branch	ALU	ALU	Multiply	Ld/St
0					
1					
2					
3					
4					

Prime Pipeline

- For this body to work, will need to setup the steady state condition for the pipeline

for (i=2;i<MAX;i++)

{ c=c+prod; prod=la*lb; la=a[i]; lb=b[i];}

Prefix

- What need to do to define the loop variables used in the pipeline?
  - prod?
  - la, lb?

for (i=2;i<MAX;i++)

{ c=c+prod; prod=la*lb; la=a[i]; lb=b[i];}

Flush Pipeline

- For this body to work, will need to finish the pipeline

for (i=2;i<MAX;i++)

{ c=c+prod; prod=la*lb; la=a[i]; lb=b[i];}

- What need to do after loop?

With Suffix

```
prod=a[0]*b[0];
la=a[1]; lb=b[1];
for (i=2;i<MAX;i++)
   { c=c+prod; prod=la*lb; la=a[i]; lb=b[i];}
c=c+prod;
c=c+la*lb;
```

Midterm

Midterm

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

Preclass 4 Revisit Day 13 Preclass 2

```
for (xptr=&x;xptr<XMAX;xptr++)
  res[i]=sqrt(x[i]*x[i]+y[i]*y[i]+z[i]*z[i]);
```

- <XMAX;i++; xptr++; yptr++; zptr++; ld x; ldy; ldz; $x[i]^2$; $y[i]^2$; $z[i]^2$; $x[i]^2+y[i]^2$; $+z[i]^2$; sqrt; res[i]
- What resources would it take to achieve each II by resource bound?

II	Ld	St	*	+	inc	sqrt
3						
2						
1						

Revisit Day 13 Preclass 2

- Schedule for II=3, II=2, II=1

Software Pipelining

- Observe
  - For cases without loop dependencies,
  - if willing to mix any number of loop instances,
  - can achieve resource bound
- May require more registers to hold state

Compare to Day 13 (no software pipelining)

- One of each: II=8
- Latency lower bound (roughly II=1 hardware here)
  - II=2.5 for unroll 4 iterations (10 cycles)
  - II=7

Lessons

- VLIW provides rich area-time tradeoffs
- Pipelining not just for hardware
  - Already seen for coarse operation pipelining, even with processors
    - Process- or thread-level parallelism
  - Now see for ILP

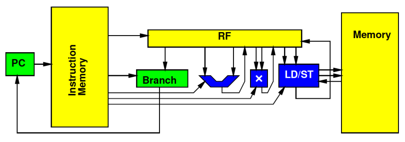
Midterm

- Closed book, notes, etc.
- Calculators allowed (encouraged)
- Last year's midterm, final online
  - Both without answers (for practice)
  - …and with answers (check yourself)
- No VLIW on midterm

Looping

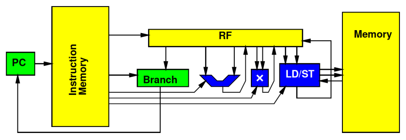
Loop Overhead

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



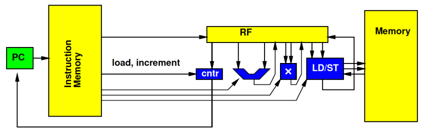
VLIW Loop Overhead

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

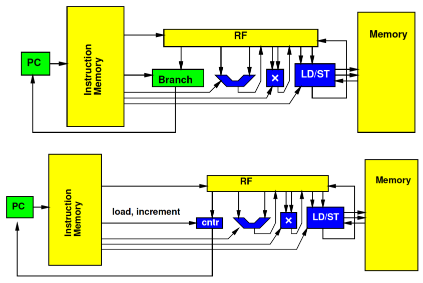


Zero-Overhead Loops

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

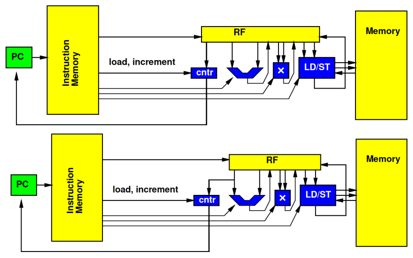


Simplification



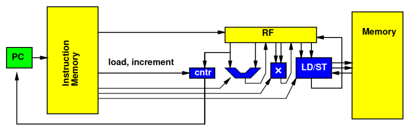
Zero-Overhead Loop Simplify

- Share port – simplify further



Zero-Overhead Loop Example (with software pipelining)

```
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 Branching

Branch Challenge

- Can only have one branch unit
  - What would mean to have two branches issued in a long instruction?
- Now have much more may do on a cycle
- Branches break up ability to schedule all the VLIW operators

Example

```
if (a==0)
  c=MAXVAL;
else
  c=a/b;
```

Addr	BR	ALU	Mpy	Div	LD/ST
0		a==0			
1	Brz 3				
2	Br 4			c=a/b	
3		c=MAX VAL			

Day 6

Predicated Operation

- Many architectures will provide a predicated operation
- Only perform operation when predicate matches instruction

- p[i]=a[i]<b[i]
- p[i]: d[i]=c[i] + a[i]
- ~p[i]: d[i]=c[i] + b[i]

Example with Predication

```
if (a==0)                p=(a==0)
  c=MAXVAL;              p: c=MAXVAL;
else                     ~p: c=a/b;
  c=a/b;
```

Addr	BR	ALU	Mpy	Div	LD/ST
0		p=(a==0)			
1		p:c=MAXVAL		~p: c=a/b	

Big Ideas:

- Pipelining of data processing useful for software-scheduled on (VLIW) processors
  - Not just hardware pipelines

Admin

- HW6 due Friday
  - Remember many slow builds
- Friday 2pm – talk on DRAM Scaling
  - Levine 307
- Midterm Monday
  - See Spring 2017 syllabus for
    - Last semesters midterm and final
      - …with solutions