

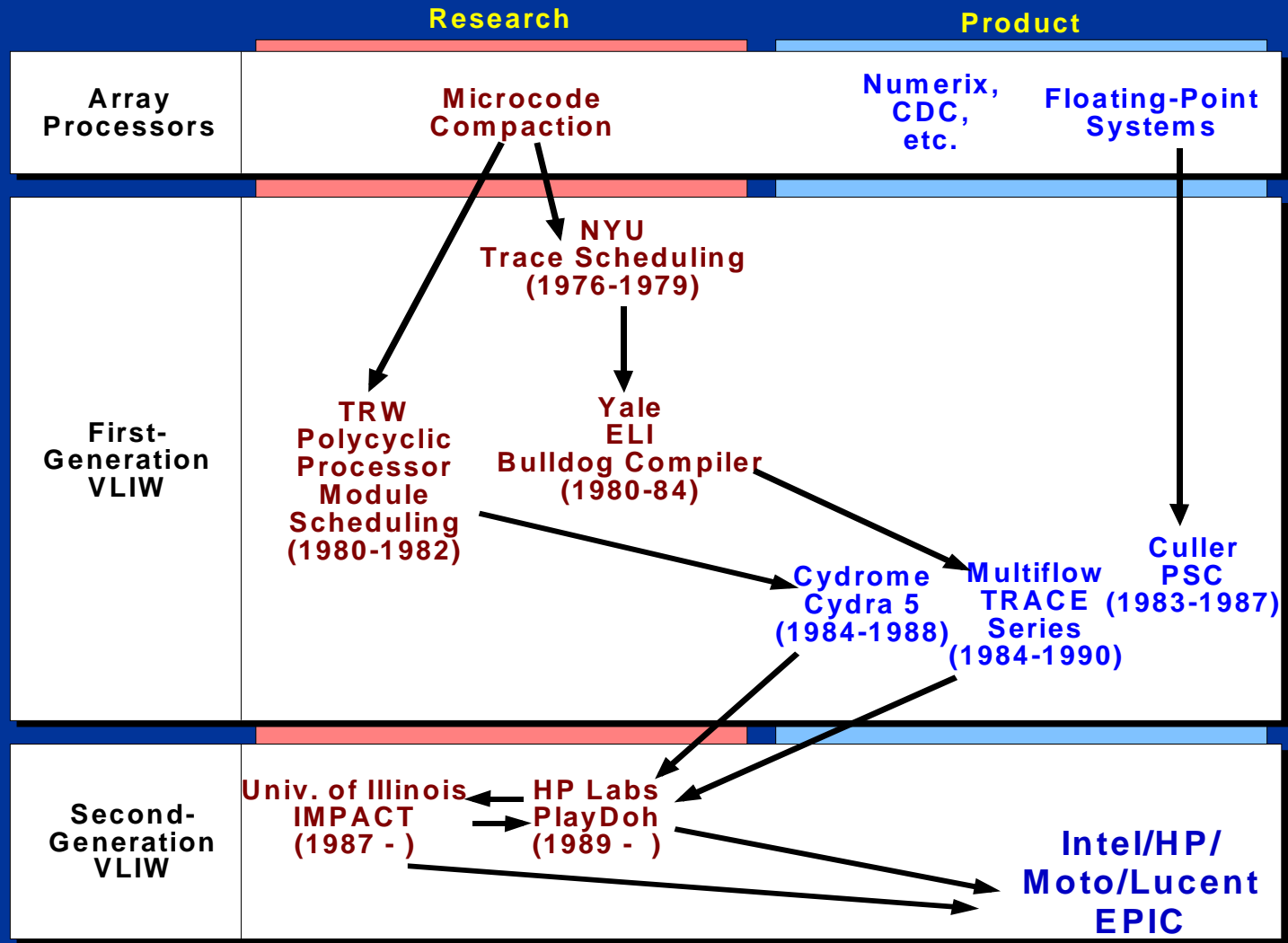
# VLIW Architecture

Wen-mei Hwu

ECE

University of Illinois,  
Urbana-Champaign

# Evolution of VLIW/EPIC



# Traditional VLIW Processor

- Defining attributes
  - MultiOp: instruction containing multiple independent operations
    - no flow dependences between these operations
    - output and anti-dependences specified by the assumed latencies
  - Exposed, architectural latencies
  - Exposed resource types and configurations

# Traditional VLIW Processor

- Advantages
  - No runtime dependence checks against previously or simultaneously issued operations
  - No runtime scheduling decisions
  - No need for register renaming

# Traditional VLIW Processor

- Disadvantages
  - No tolerance for different or variable latencies
  - No tolerance for any difference in the types of functional units
  - No object code compatibility

# Unit assumed latency

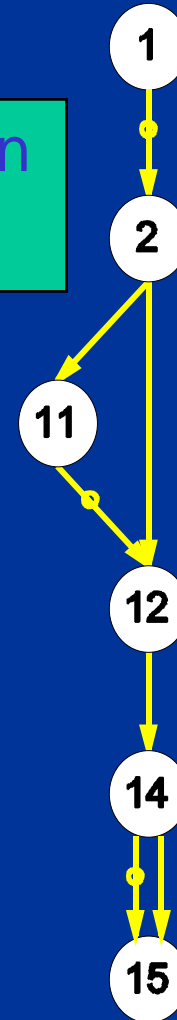
- A conventional sequential program has unit assumed latency (UAL)
  - The semantics of the program are understood by assuming each instruction is completed before the next one is issued, i.e. value visibility

<b>f3 = f4 x f5</b>	<b>r6 = r7 + r8</b>	<b>r1 = load(r2)</b>	<b>beq(f4,f5)</b>
<b>MultiOp + UAL</b>			

# UAL Dependence Semantics

<u>Instruction</u>	<u>Operation</u>
1	$r1 = \text{load}(r2)$
2	$r1 = \text{load}(r3)$
3	
4	
5	
6	
7	
8	
9	
10	
11	$r4 = \text{fmul}(r1, r5)$
12	$r4 = \text{fadd}(r1, r6)$
13	
14	$r7 = \text{fmul}(r4, r9)$
15	$r7 = \text{fadd}(r7, r8)$

Execution  
timing

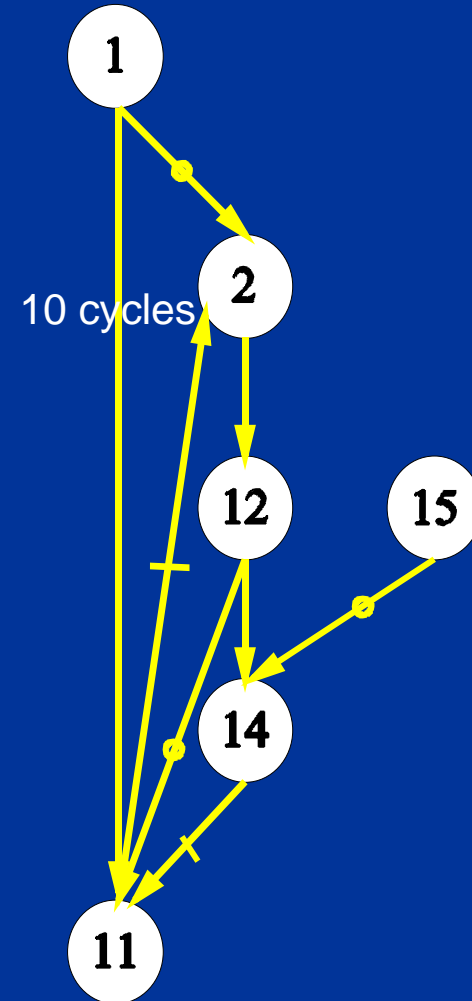


# Non-unit assumed latency

- Program has non-unit assumed latency (NUAL) if:
  - At least one operation has a non-unit assumed latency,  $L$ , which is greater than 1
  - The semantics of the program are correctly understood if exactly the next  $L-1$  instructions are understood to have issued before this operation completes

# NUAL Dependence Semantics

<u>Instruction</u>	<u>Phase1 Operation</u>	<u>Phase2 Operation</u>
1	v1 = load(r2)	
2	v2 = load(r3)	
3		
4		
5		
6		
7		
8		
9		
10		r1 = v1
11	v3 = fmul(r1, r5)	r1 = v2
12	v4 = fadd(r1, r6)	
13		r4 = v4
14	v5 = fmul(r4, r9)	r4 = v3
15	v6 = fadd(r7, r8)	
16		r7 = v6
17		r7 = v5



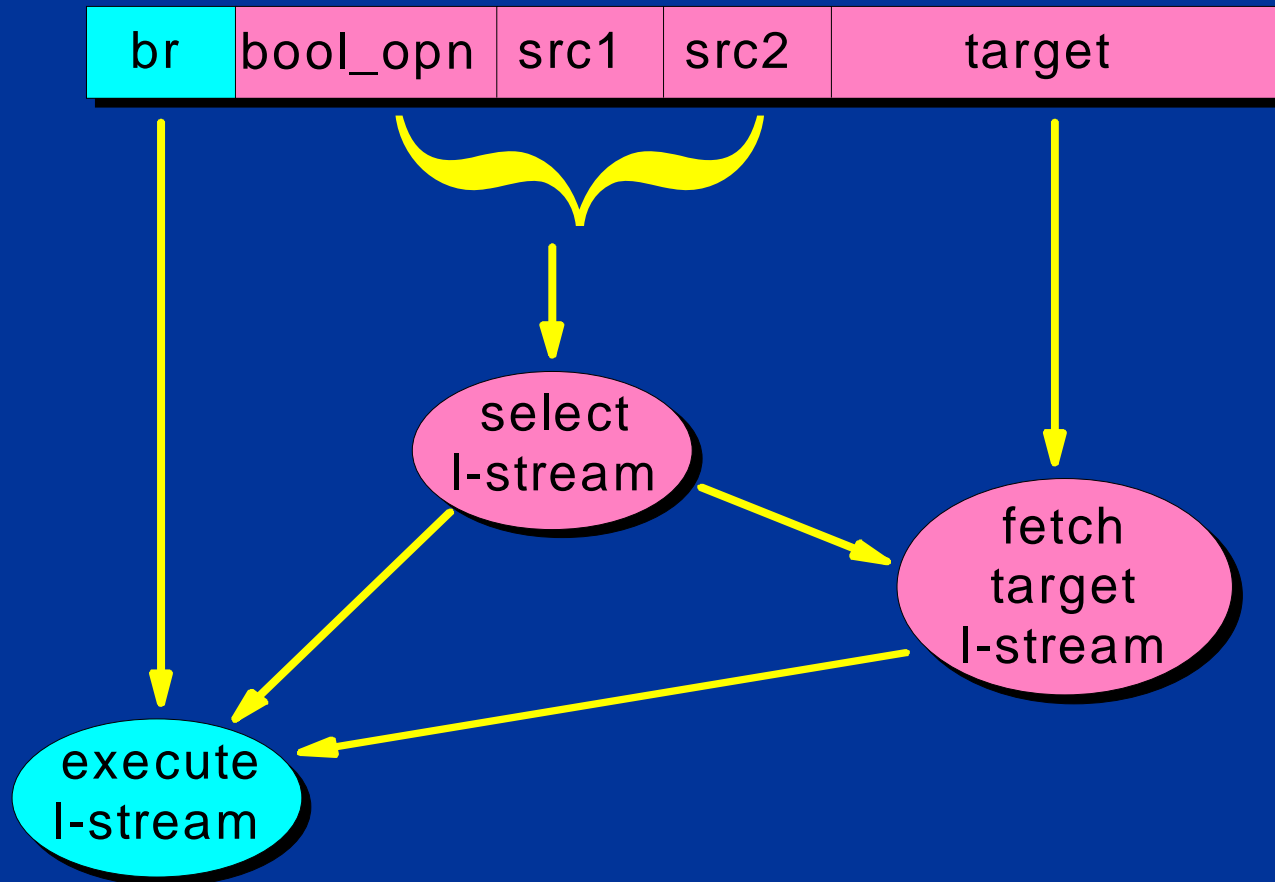
# NUAL scheduling models

- Equals (EQ) Model
  - Each operation takes exactly its specified latency
    - i.e. the destination register will not be written until latency number of cycles
  - More efficient register usage
  - No need for register renaming or buffering
    - Bypass from FU output to inputs
    - Register writes whenever FU completes

# NUAL Scheduling Models

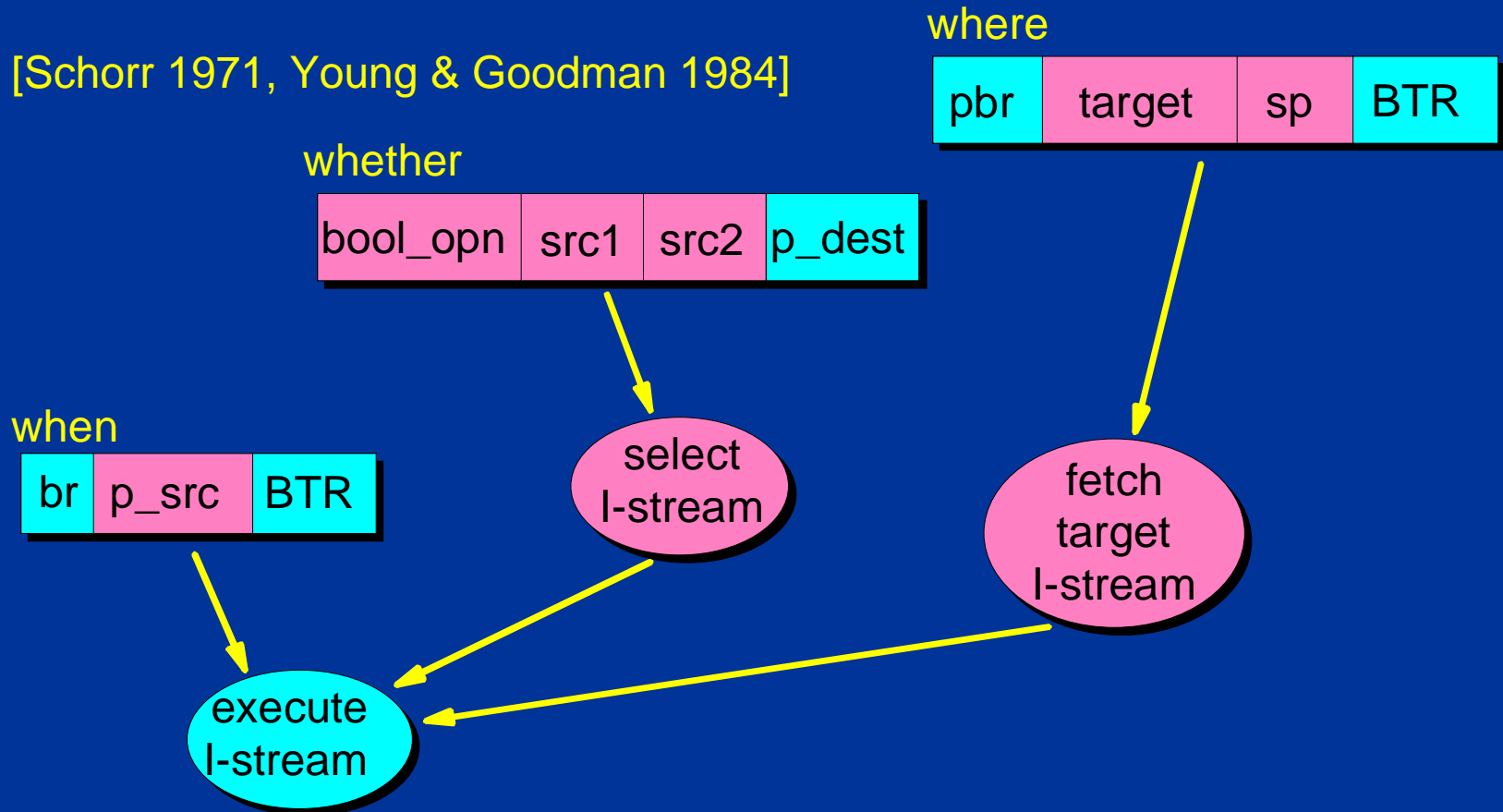
- Less-Than-or-Equals (LEQ) Model
  - An operation may take less than or equal to its specified latency
    - i.e. the destination register can be written any time from issue to latency cycles
    - Dependent operation still needs to be scheduled at or after latency
  - Simplifies the implementation of precise interrupts
  - Provides binary compatibility when latencies are reduced

# Conventional conditional branch



# Unbundled conditional branch

[Schorr 1971, Young & Goodman 1984]



# Reduced Branch Latency

- Early calculation of branch target
  - Writing the target of the branch into a branch target register conveys branch target to the instruction fetch unit
  - If the target address is available at fetch unit early enough, the lcache latency is effectively reduced to 0

# Reduced Branch Latency

- Early determination of the branch condition
  - At the time the branch operation is encountered, it is known whether the branch is to be taken or not
  - If the source predicate is known at issue time, the branch latency is effectively reduced to 0

# Code Size Considerations

- Traditional VLIW require nops to fill unscheduled slots
  - Double penalty for poor schedule: execution cycles and code size
  - Larger code size in general
- Modern EPIC processors deal with this deficiency with compaction encoding

# Example TINKER Encoding (Conte, et al)

- Variable-width MultiOp
- Fixed-width Ops (64-bits)
  - Header bit (cycle starting with this operation), optype (dispersement), pause specifier (cycles of nops inserted after current cycle)
- Precursor to Itanium template encoding

# Example: Tinker Encoding

(See fig.)

