# CSE 520
# Computer Architecture II

## Complex Pipelining: VLIW

Prof. Michel A. Kinsy

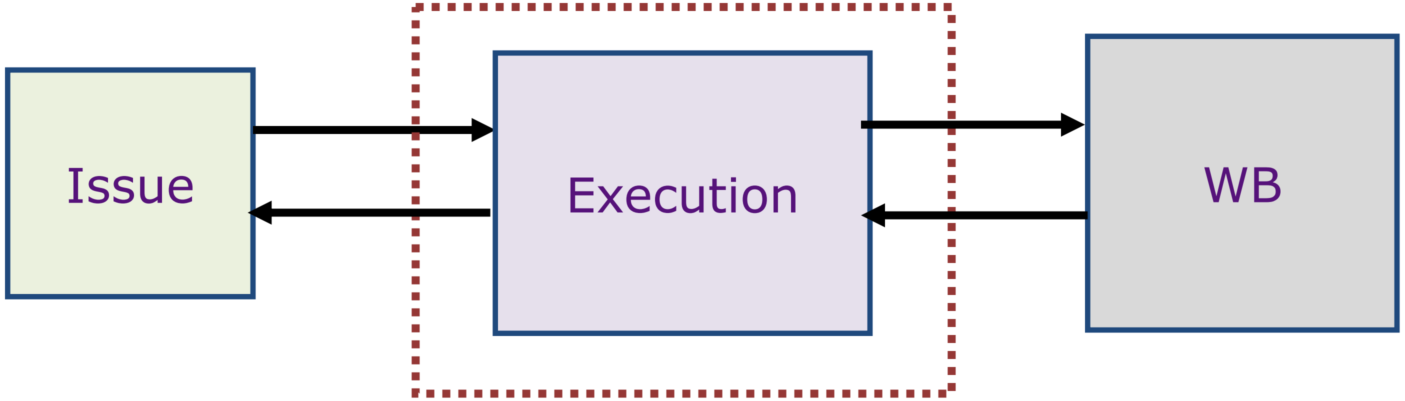
# Execution Concurrency Limits

- Which features of an ISA limit the number of instructions in the pipeline?
  - Number of Registers

- Which features of a program limit the number of instructions in the pipeline?
  - Control transfers

# Little's Law

- Throughput (T) = Number in Flight (N) / Latency (L)



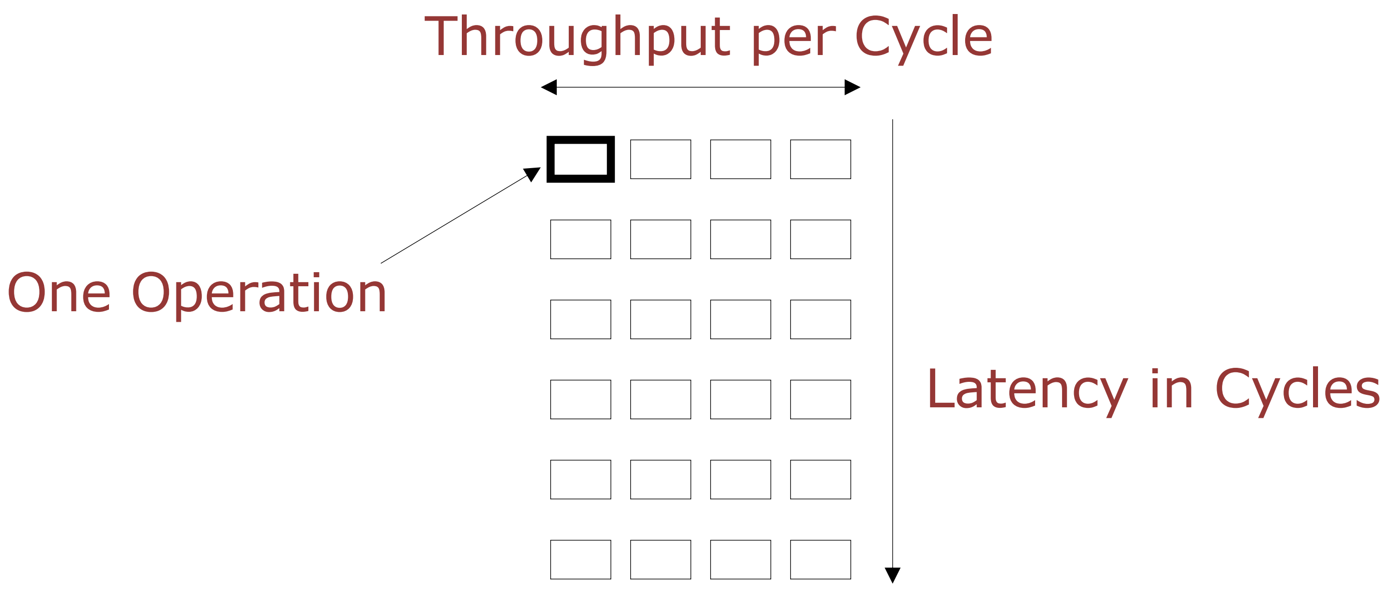

- Illustrative Example
  - 4 floating point units
  - 8 cycles per floating point operation
    - 1/2 issues per cycle!

# Little’s Law

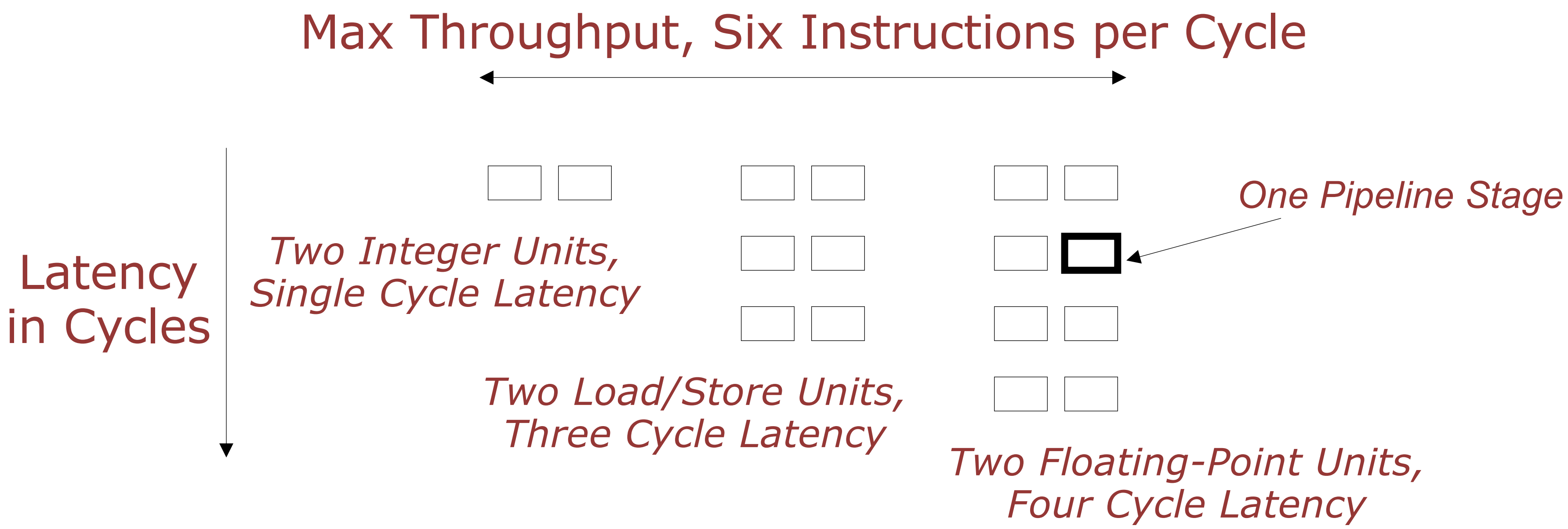
*Parallelism = Throughput * Latency*

*or*

$$\overline{N} = \overline{T} \times \overline{L}$$



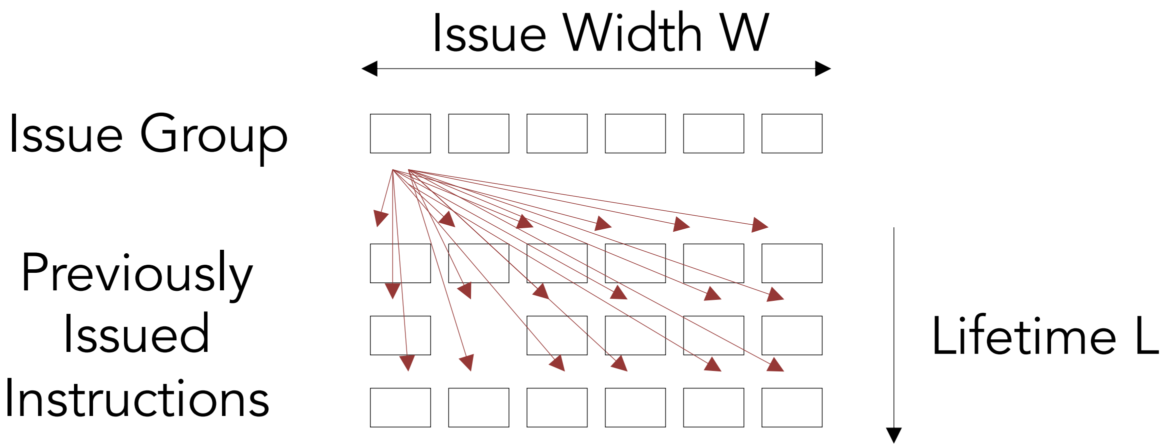
# Pipelined ILP Machine




- How much instruction-level parallelism (ILP) required to keep machine pipelines busy?

# Superscalar Control Logic Scaling

- Each issued instructions must make interlock checks against W*L instructions, i.e., growth in interlocks $\propto$ W*(W*L)
- For in-order machines, L is related to pipeline latencies
- For out-of-order machines, L also includes time spent in instruction buffers (instruction window or ROB)
- As W increases, larger instruction window is needed to find enough parallelism to keep machine busy  greater L
  - Out-of-order control logic grows faster than $W^2$ ($\sim W^3$)



# Superscalar Execution

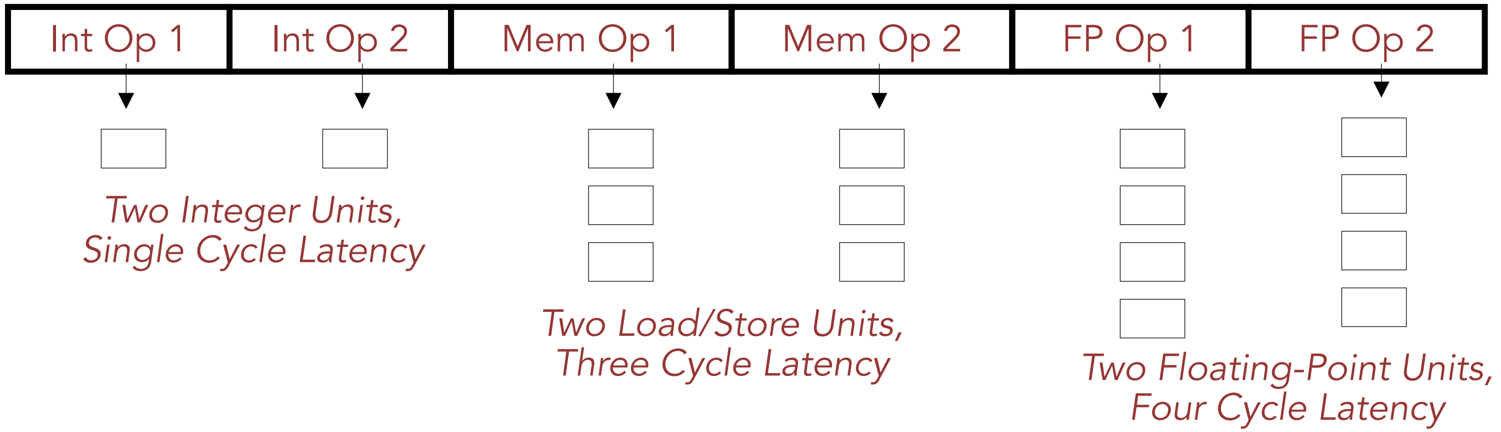
- Receive conventional instructions conceived for sequential processors
- Parallel Execution
  - Done dynamically at run-time by the hardware
  - Data dependency is checked and resolved in hardware
  - Need a look-ahead hardware window for instruction fetch

# VLIW: Very Long Instruction Word

- A very long instruction word consists of multiple independent instructions packed together by the compiler
  - Packed instructions can be logically unrelated (contrast with SIMD)
- Idea: Compiler finds independent instructions and statically schedules (i.e. packs/bundles) them into a single VLIW instruction
- Traditional Characteristics
  - Multiple functional units
  - Each instruction in a bundle executed in lock step
  - Instructions in a bundle statically aligned to be directly fed into the functional units

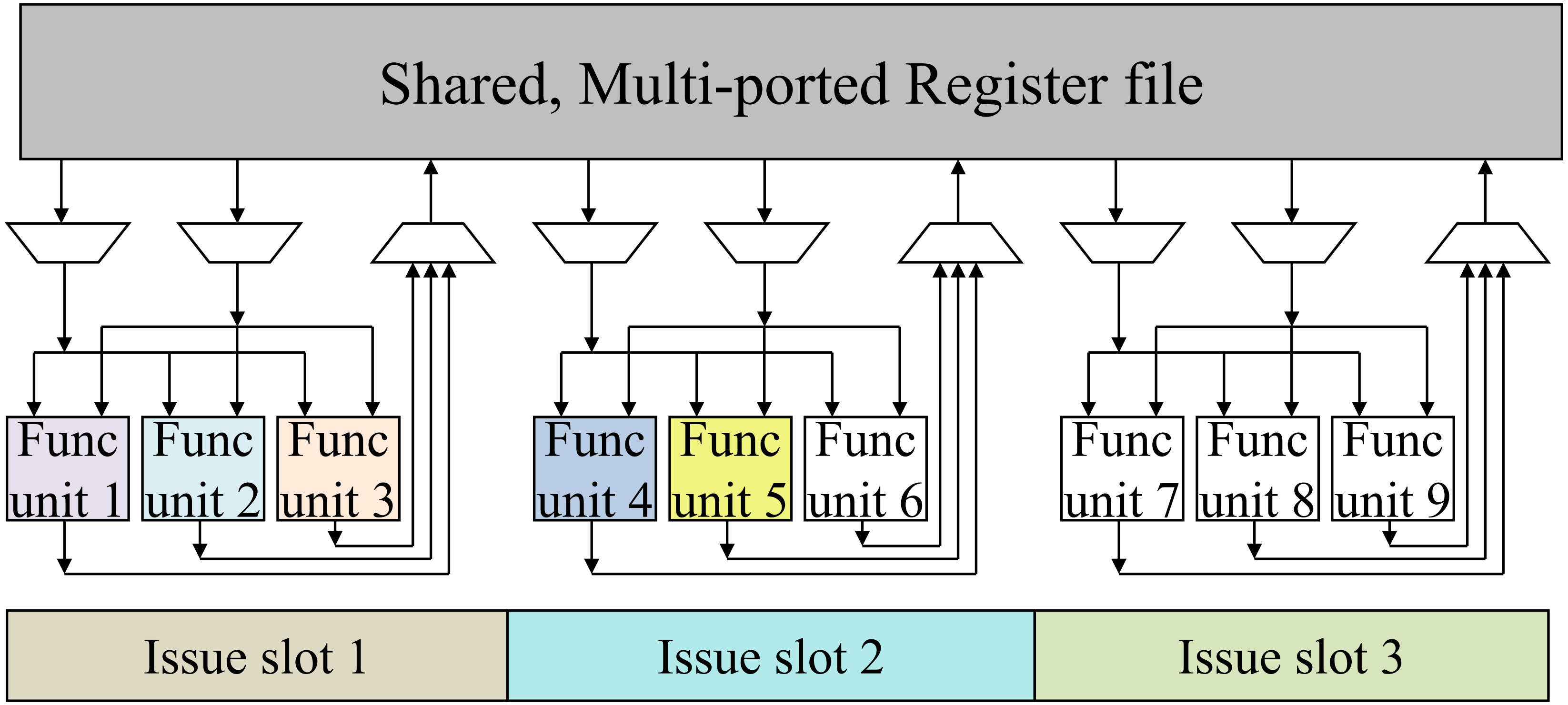
# VLIW: Very Long Instruction Word

- Multiple operations packed into one instruction
- Each operation slot is for a fixed function
- Constant operation latencies are specified
- Architecture requires guarantee of:
  - Parallelism within an instruction  no x-operation RAW check
  - No data use before data ready  no data interlocks



# VLIW: Very Long Instruction Word

| | | | | | |
|---|---|---|---|---|---|
| field → | FU 1 | FU 2 | FU 3 | FU 4 | FU 5 |
| instruction → | subi x8, x5, 3 | andi x1, x5, 12 | mul x6, x5, x2 | lw x3, 0(x5) | bnez x5, Exit |



# VLIW: Very Long Instruction Word

- Static scheduling done at compile-time by the compiler
- Advantages
  - No need for dynamic scheduling hardware
  - No need for dependency checking within a VLIW instruction
  - No need for instruction alignment/distribution after fetch to different functional units
    - Reduce hardware complexity
    - Tasks such as decoding, data dependency detection, instruction issue, …, etc. becoming simple
    - Potentially higher clock rate
    - Higher degree of parallelism with global program information

# VLIW Compiler

- The compiler:
  - Schedules to maximize parallel execution
  - Guarantees intra-instruction parallelism
  - Schedules to avoid data hazards (no interlocks)
    - Typically separates operations with explicit NOP
- Higher complexity of the compiler
    - Compiler optimization needs to consider technology dependent parameters such as latencies and load-use time of cache.
    - Non-deterministic problem of cache misses, resulting in worst case assumption for code scheduling
    - In case of un-filled opcodes in a (V)LIW, memory space and instruction bandwidth are wasted

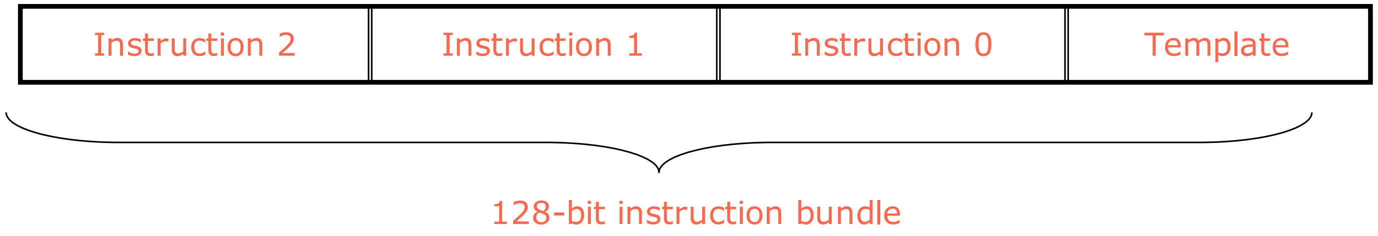
# Early VLIW Machines

- FPS AP120B (1976)
  - Scientific attached array processor
  - First commercial wide instruction machine
- Multiflow Trace (1987)
  - Available in configurations with 7, 14, or 28 operations/instruction
  - 28 operations packed into a 1024-bit instruction word
- Cydrome Cydra-5 (1987)
  - 7 operations encoded in 256-bit instruction word
  - Rotating register file

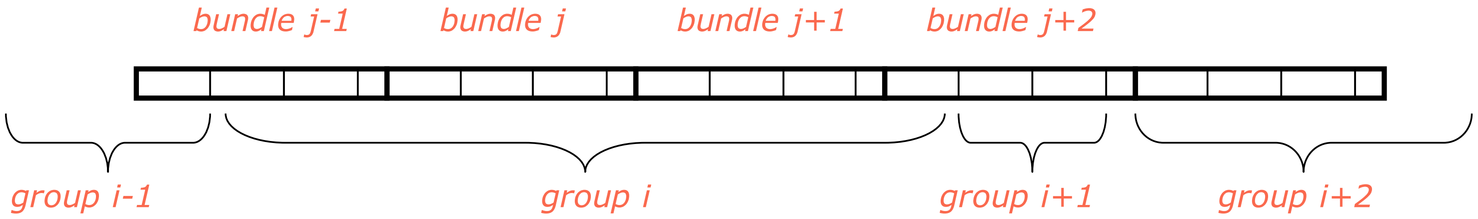
# Intel EPIC IA-64

- EPIC is the style of architecture
  - Explicitly Parallel Instruction Computing
- IA-64 is Intel's chosen ISA
  - IA-64 = Intel Architecture 64-bit
  - An object-code compatible VLIW
- Itanium (aka Merced) is first implementation (cf. 8086)
  - First customer shipment expected 1997 (actually 2001)
  - McKinley, second implementation shipped in 2002

# IA-64 Instruction Format




- Template bits describe grouping of these instructions with others in adjacent bundles
- Each group contains instructions that can execute in parallel



# Problems with "Classic" VLIW

- Knowing branch probabilities
  - Profiling requires an significant extra step in build process
- Object code size
  - Instruction padding wastes instruction memory/cache
  - Loop unrolling/software pipelining replicates code
- Scheduling variable latency memory operations
  - Caches and/or memory bank conflicts impose statically unpredictable variability

# Problems with “Classic” VLIW

- Scheduling for statically unpredictable branches
  - Optimal schedule varies with branch path
- Object-code compatibility
  - Have to recompile all code for every machine, even for two machines in same generation

# Loop Unrolling

- Unroll inner loop to perform 4 iterations at once
  - Is this code correct?

```
for (i=0; i<N; i++)
    B[i] = A[i] + C;
```

→

```
for (i=0; i<N; i+=4)
{
    B[i]     = A[i] + C;
    B[i+1] = A[i+1] + C;
    B[i+2] = A[i+2] + C;
    B[i+3] = A[i+3] + C;
}
```

# VLIW Summary

- VLIW simplifies hardware, but requires complex compiler techniques
- Solely-compiler approach of VLIW has several downsides that reduce performance
  - Too many NOPs (not enough parallelism discovered)
  - Static schedule intimately tied to microarchitecture
  - Code optimized for one generation performs poorly for next
  - No tolerance for variable or long-latency operations (lock step)
- Most compiler optimizations developed for VLIW employed in optimizing compilers (for superscalar compilation)
  - Enable code optimizations
  - VLIW successful in embedded markets, e.g. DSP

# Superscalar and VLIW Machines

- Superscalar architecture implements instruction-level parallelism
  - Single Instructions-Single Data (SISD) format
- VLIW machines show the advantages and limitations of instruction-level parallelism
  - Multiple Instructions-Multiple Data (MIMD)
- We will further explore MIMD types of execution with multicore processors
- Single Instructions-Multiple Data (SIMD) execution with vector processor

# Next Class

- SIMD and Vector Processors