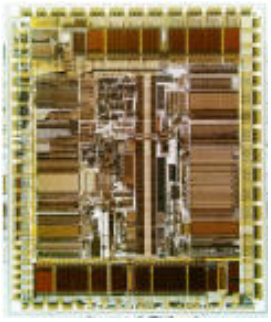
# Lecture 13: Branch Prediction & Instruction Level Parallelism

- Administrative:
  - Partial solutions to HW #4 out… Including bug-fix to HW #4
  - Return written portion of HW #2
- Last Time:
  - Pipeline hazards: Data, Control, Structural
  - Fixes: Bypassing/forwarding
- Today
  - Branch prediction (on blackboard)
  - Static multiple-issue (slides)
  - Dynamic, out-of-order multiple issue (on blackboard)



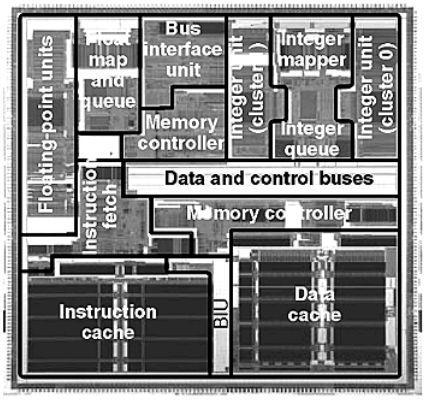
# Where Are We?

- Pipelined in-order processor
- Simple branch prediction
- Instruction/data caches (on –chip)



DEC Alpha 21064

- Out-of-order instruction execution
- "Superscalar"
- Sophisticated branch prediction




DEC Alpha 21264

Branch prediction blackboard notes
(see separate text file)



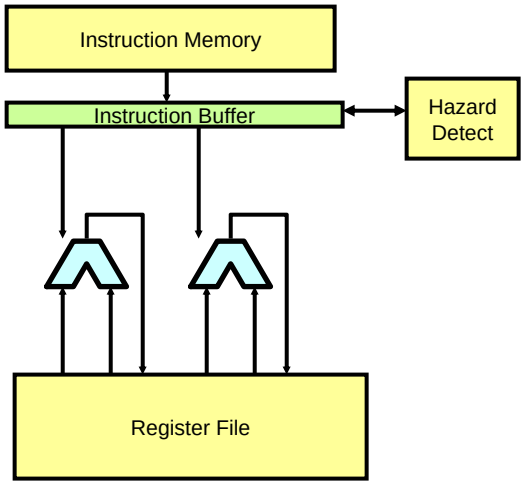
## How Do We Speed up the Pipeline?

- Instruction Level Parallelism (ILP)
  - Multi-issue (in-order execution to multiple pipes)
  - Dynamic scheduling ("Superscalar")
  - Compiler schedules ILP (VLIW/EPIC)
- Undetermined dependencies at compile time ⇒ dynamic scheduling
  - Object code compatibility
  - Simplify compiler
- WAR/WAW hazards ⇒ register renaming

```
ADD R1,R2,R3          ADD R1,R2,R3
SUB R1,R4,R5    ⇒     SUB R1',R4,R5
```

- Too many branches ⇒ better branch prediction
  - Or use predication to eliminate branches
- Unknown dependencies (control/data) ⇒ speculate

## Multiple Issue – No instruction reordering






## Multiple Issue (Details)

- Dependencies and structural hazards checked at run-time
- Can run existing binaries
  - Recompiler for performance, not correctness
  - Example - Pentium
- More complex issue logic
  - Swizzle next N instructions into position
  - Check dependencies and resource needs
  - Issue M <= N instructions that can execute in parallel

# Example Multiple Issue

Issue rules: at most 1 load/store, at most 1 floating op

Latency: load=1, int=1, float-mult = 1, float-add = 1

```
                                                        cycle
LOOP:  LD      F0, 0(R1)       // a[i]                  1
       LD      F2, 0(R2)       // b[i]                  2
       MULTD   F8, F0, F2      // a[i] * b[i]           4 (stall)
       ADDD    F12, F8, F16    // + c                   5
       SD      F12, 0(R3)      // d[i]                  6
       ADDI    R1, R1, 4
       ADDI    R2, R2, 4                                7
       ADDI    R3, R3, 4
       ADDI    R4, R4, 1       // increment I            8
       SLT     R5, R4, R6      // i<n-1                  9
       BNEQ    R5, R0, LOOP                           10
```

**Old CPI = 12/11 = 1.09**
**New CPI = 10/11 = 0.91**



# Rescheduled for Multiple Issue

Issue rules: at most 1 LD/ST, at most 1 floating op

Latency: LD - 1, int-1, F*-1, F+-1

```
                                                        cycle
LOOP:  LD      F0, 0(R1)       // a[i]                  1
       ADDI    R1, R1, 4
       LD      F2, 0(R2)       // b[i]                  2
       ADDI    R2, R2, 4
       MULTD   F8, F0, F2      // a[i] * b[i]           4
       ADDI    R4, R4, 1       // increment I
       ADDD    F12, F8, F16    // + c                   5
       SLT     R5, R4, R6      // i<n-1
       SD      F12, 0(R3)      // d[i]                  6
       ADDI    R3, R3, 4
       BNEQ    R5, R0, LOOP                           7
```

**Old CPI = 0.91**
**New CPI = 7/11 = 0.64**

## The Problem with Static Scheduling

- In-order execution
  - an unexpected long latency blocks ready instructions from executing
  - binaries need to be rescheduled for each new implementation
  - small number of *named* registers becomes a bottleneck

```
LW    R1, C    //miss 50 cycles
LW    R2, D
MUL   R3, R1, R2
SW    R3, C
LW    R4, B    //ready
ADD   R5, R4, R9
SW    R5, A
LW    R6, F
LW    R7, G
ADD   R8, R6, R7
SW    R8, E
```



## Dynamic Scheduling

- Determine execution order of instructions at *run time*
- Schedule with knowledge of run-time variable latency
  - cache misses
- Compatibility advantages
  - avoid need to recompile old binaries
  - avoid bottleneck of small *named* register sets
    - but still need to deal with spills
- Significant hardware complexity

## Summary

- Today:
  - Branch prediction – static and dynamic
  - Static multiple issue – VLIW and Superscalar
  - Dynamic multiple issue – basic idea only.
- Next Time
  - Caches (new topic!)
  - Read: P&H 7.1 – 7.3