# ECE 4750 Computer Architecture, Fall 2015

# T11 Advanced Processors: Register Renaming

School of Electrical and Computer Engineering
Cornell University

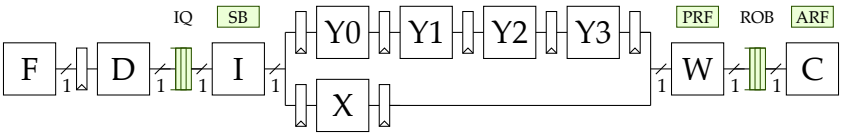
revision: 2015-11-11-13-08

# 1. WAW and WAR Hazards

```
a: mul    r1,   r2,   r3
b: mul    r4,   r1,   r5
c: addiu  r6,   r4,   1
d: addiu  r4,   r7,   1
```

- RAW data hazards vs. WAW/WAR name hazards
  - RAW dependencies are "true" data dependencies because we actually pass data from the writer ot the reader
  - WAW/WAR dependencies are not "true" data dependencies
  - WAW/WAR dependencies exist because of limited "names"
  - Can always avoid WAW/WAR hazards by renaming registers in software, but eventually we will run out of register names
  - Key Idea: Provide more "physical registers" and rename architectural to physical registers in hardware

## WAW/WAR name hazards in IO2L microarchitecture

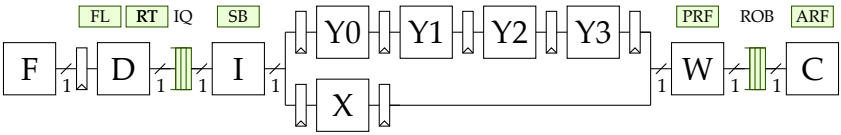



| | | | | |
|---|---|---|---|---|
| **ARF** | | | | write |
| **PRF** | | read | write | read |
| **SB** | | read/write | | |
| **IQ** | alloc | read/dealloc | | |
| **ROB** | alloc | | write | read/dealloc |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a : `mul r1, r2, r3` | | | | | | | | | | | | | | | |
| b : `mul r4, r1, r5` | | | | | | | | | | | | | | | |
| c : `addiu r6, r4, 1` | | | | | | | | | | | | | | | |
| d : `addiu r4, r7, 1` | | | | | | | | | | | | | | | |

- Explore two different schemes
  - Store pointers in the IQ and ROB
  - Store values in the IQ and ROB
- For each scheme
  - overall pipeline structure
  - required hardware data-structures
  - example instruction sequence executing on microarchitecture
- Several simplifications
  - all designs are single issue
  - only support `addu`, `addiu`, `mul`

# 2. IO2L Pointer-Based Register Renaming Scheme




| | | | | |
|---|---|---|---|---|
| **ARF** | | | | write |
| **PRF** | | read | write | |
| **SB** | | read/write | | |
| **IQ** | alloc | read/dealloc | | |
| **ROB** | alloc | | write | read/dealloc |
| **FL** | read/alloc | | | dealloc |
| **RT** | read/write | | write | |

- Increase the size of the PRF to provide more "names"
- Add free list (FL) in D stage
  - FL holds list of unallocated physical registers
  - Physical registers allocated in D and deallocated in C
- Add rename table (RT) in D stage
  - RT maps architectural registers to physical registers
  - Sometimes called the "map table"
  - Destination register renamed in D stage
  - Look up renamed source registers in D, and write these physical register specifiers into the IQ
- Modify SB and ROB
  - Scoreboard indexed by physical reg instead of architectural reg
- NOTE: Values can only be bypassed or read from the PRF
- I/X/Y/W stages only manipulate physical registers

## Data Structures: FL, RT, Modified ROB

**Rename Table**

| | p | preg |
|---|---|---|
| r1 | 1 | p0 p7 |
| r2 | 0 | p1 |
| r3 | 0 | p2 |
| r4 | 1 | p3 p8 |
| r5 | 0 | p4 |
| r6 | 1 | p5 p9 |
| r7 | 0 | p6 |
| ... | | ... |
| r31 | 0 | p63 |

**Free List**

| | free? |
|---|---|
| p0 | 0 |
| p1 | 0 |
| ... | ... |
| p8 | 0 |
| p9 | 0 |
| p10 | 1 |
| p11 | 1 |
| ... | ... |
| p63 | 0 |

**Reorder Buffer**

| v | p | v | preg | areg | ppreg |
|---|---|---|---|---|---|
| 1 | 1 | 1 | p7 | p8 | p9 |
| 1 | 1 | 1 | p8 | p4 | p3 |
| 1 | 1 | 1 | p9 | p6 | p5 |
| 0 | -- | -- | -- | -- | -- |

- Free List (FL)
  - **free**: one if corresponding preg is free
  - Use priority encoder to allocate first free preg
- Rename Table (RT)
  - **p**: pending bit, is a write to this areg in flight?
  - **preg**: what preg the corresponding areg maps to
  - Entries in RT are always valid
- Modified Reorder Buffer (ROB)
  - Include three fields with pointers to PRF and ARF
  - **preg**: pointer to register in PRF that holds result value
  - **areg**: pointer to register in ARF to copy value into
  - **ppreg**: pointer to previous register in PRF for this areg

Can only free a physical register when we can guarantee no reads of that physical register are still in flight!

## Example Execution Diagrams

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| `a: mul r1, r2, r3` | | | | | | | | | | | | | | | |
| `b: mul r4, r1, r5` | | | | | | | | | | | | | | | |
| `c: addiu r6, r4, 1` | | | | | | | | | | | | | | | |
| `d: addiu r4, r7, 1` | | | | | | | | | | | | | | | |

**Issue Queue**

| op | v | imm | v | dest | v | p | src0 | v | p | src1 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |

**Rename Table**

| | p | preg |
|---|---|---|
| r1 | | p0 |
| r2 | | p1 |
| r3 | | p2 |
| r4 | | p3 |
| r5 | | p4 |
| r6 | | p5 |
| r7 | | p6 |
| ... | | ... |
| r31 | | |

**Free List**

| | free? |
|---|---|
| p0 | |
| p1 | |
| p2 | |
| p3 | |
| p4 | |
| p5 | |
| p6 | |
| p7 | 1 |
| p8 | 1 |
| p9 | 1 |
| p10 | 1 |

**Physical Register File**

| | value |
|---|---|
| p0 | |
| p1 | 1 |
| p2 | 2 |
| p3 | |
| p4 | 4 |
| p5 | |
| p6 | 5 |
| p7 | ... |
| p8 | |
| p9 | |
| p10 | |

**Architectural Register File**

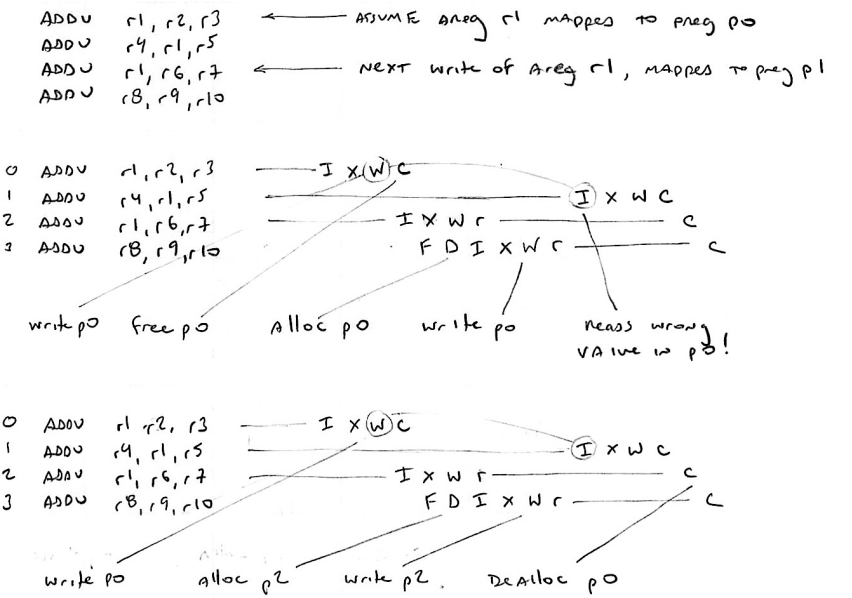
| | value |
|---|---|
| r1 | |
| r2 | 1 |
| r3 | 2 |
| r4 | |
| r5 | 4 |
| r6 | |
| r7 | 5 |
| ... | ... |
| r31 | |

**Reorder Buffer**

| p | v | preg | areg | ppreg |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |

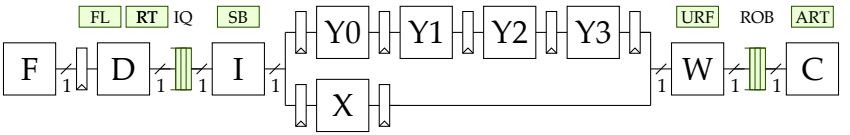
| | | | | | Rename Table | | | | | | | | Free List | Issue Queue | | | | Reorder Buffer | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Cycle | D | I | W | C | r1 | r2 | r3 | r4 | r5 | r6 | r7 | | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 |
| 0 | | | | | p0 | p1 | p2 | p3 | p4 | p5 | p6 | p7, p8, p9, p10 | | | | | | | | |
| 1 | a | | | | — | — | — | — | — | — | — | p7, p8, p9, p10 | | | | | | | | |
| 2 | b | a | | | p7* | — | — | — | — | — | — | p8, p9, p10 | p7/p1/p2 | | | | | | | |
| 3 | c | | | | — | — | — | p8* | — | — | — | p9, p10 | | p8/p7/p4* | | | p7*/r1/p0 | | | |
| 4 | d | | | | — | — | — | — | — | p9* | — | p10 | | — | p9/p8* | | — | — | p8*/r4/p3 | |
| 5 | | | | | — | — | — | p10* | — | — | — | | | — | — | p10/p6 | — | — | — | p9*/r6/p5 |
| 6 | | b | | | — | — | — | — | — | — | — | | | • | — | — | — | — | — | p10*/r4/p8 |
| 7 | | | a | | — | — | — | — | — | — | — | | | | — | — | — | — | — | — |
| 8 | | d | | a | p7 | — | — | — | — | — | — | | | | — | • | p7/r1/p0 | — | — | — |
| 9 | | | d | | — | — | — | — | — | — | — | p0 | | | — | | | — | — | — |
| 10 | | c | | | — | — | — | p10 | — | — | — | p0 | | | • | | | — | — | p10/r4/p8 |
| 11 | | | b | | — | — | — | — | — | — | — | p0 | | | | | | — | — | — |
| 12 | | | c | b | — | — | — | — | — | — | — | p0 | | | | | | p8/r4/p3 | — | — |
| 13 | | | | c | — | — | — | — | — | p9 | — | p0, p3 | | | | | | | p9/r6/p5 | — |
| 14 | | | | d | — | — | — | — | — | — | — | p0, p3, p5 | | | | | | | | • |
| 15 | | | | | — | — | — | — | — | — | — | p0, p3, p5, p8 | | | | | | | | |

## Freeing Physical Registers




if Areg ri is MAPPED to Preg pj, we can Free pj when the NEXT INSTRUCTION THAT writes ri COMMITS

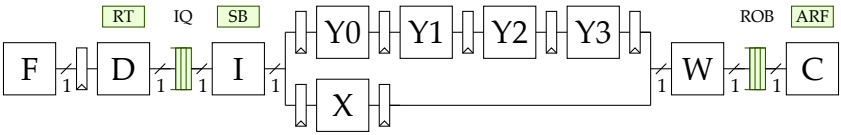
## Unified Physical/Architectural Register File




| | | | | |
|---|---|---|---|---|
| **ART** | | | | write |
| **URF** | | read | write | |
| **SB** | | read/write | | |
| **IQ** | alloc | read/dealloc | | |
| **ROB** | alloc | | write | read/dealloc |
| **FL** | read/alloc | | | dealloc |
| **RT** | read/write | | write | |

- Combine the PRF and ARF into one large unified register file (URF)
- Replace ARF with an architectural rename table (ART)
- Instead of copying *values*, C stage simply copies the preg pointer into the appropriate entry of the ART
- URF can be smaller than area for separate PRF/ARF
- Sometimes in the literature URF is just called PRF (and there is no "real" ARF, just the ART)

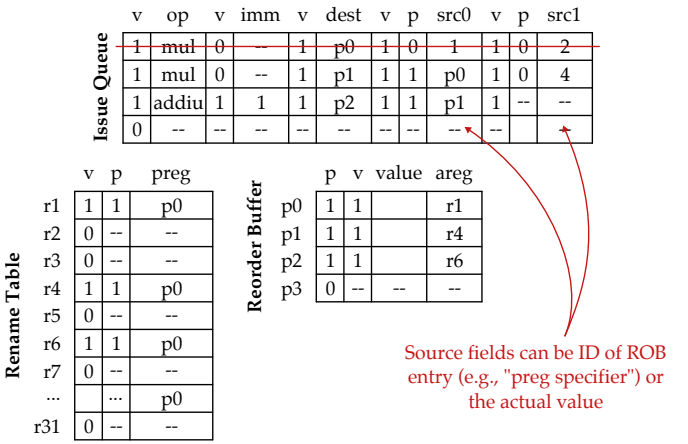
# 3. IO2L Value-Based Register Renaming Scheme




| | | | | | |
|---|---|---|---|---|---|
| **ARF** | | | | | write |
| **SB** | | read/write | | | |
| **IQ** | alloc | read/dealloc | | | |
| **ROB** | alloc | | | write | read/dealloc |
| **RT** | read/write | | | write | |

- Instead of storing future values in a separate PRF, we store them these future values in the actual ROB
- No need for FL, since "physical registers" are now really ROB entry IDs and managed naturally through ROB allocation/deallocation
- Add rename table (RT) in D stage
  - RT maps architectural registers to physical registers
  - Registers renamed in D stage, entries cleared in C
  - Destination register renamed in D stage
  - Look up renamed source registers in D, and write these physical register specifiers into the IQ
- Modify scoreboard, IQ, ROB
  - Scoreboard indexed by preg instead of areg
- NOTE: Values can be bypassed or read from either the ROB or ARF
- I/X/Y/W stages only manipulate physical registers

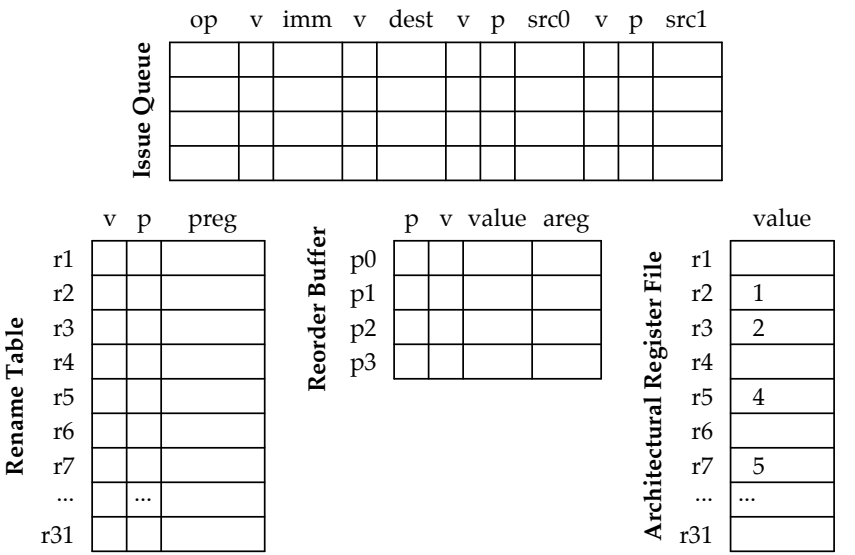
## Data Structures: RT, Modified IQ, ROB




- Rename Table (RT)
  - **v**: valid bit
  - **p**: pending bit, is a write to this areg in flight?
  - **preg**: what preg the corresponding areg maps to
  - Entries are only valid if instruction is in-flight
  - Valid bit is cleared after instruction has committed
- Modified Issue Queue (IQ)
  - **src0/src1**: when pending bit is set, source fields contain the preg specifier (i.e., ROB entry ID) that we are waiting on; when pending bit is clear, source fields contain the *values*
- Modified Reorder Buffer (ROB)
  - Replace single rdest field with two new fields
  - **value**: actual result value
  - **areg**: pointer to register in ARF to copy value into

## Example Execution Diagrams

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a : `mul r1, r2, r3` | | | | | | | | | | | | | | | |
| b : `mul r4, r1, r5` | | | | | | | | | | | | | | | |
| c : `addiu r6, r4, 1` | | | | | | | | | | | | | | | |
| d : `addiu r4, r7, 1` | | | | | | | | | | | | | | | |



We can use a table to compactly illustrate how IO2L value-based register renaming works. We show the state of the RT and ROB at the beginning of every cycle.

| | | | | | Rename Table | | | | | | | Issue Queue | | | | Reorder Buffer | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Cycle** | **D** | **I** | **W** | **C** | **r1** | **r2** | **r3** | **r4** | **r5** | **r6** | **r7** | **0** | **1** | **2** | **3** | **0** | **1** | **2** | **3** |
| 0 | | | | | | | | | | | | | | | | | | | |
| 1 | a | | | | | | | | | | | | | | | | | | |
| 2 | b | a | | | p0* | | | | | | | p0/r2/r3 | | | | p0*/r1 | | | |
| 3 | c | | | | \| | | | p1* | | | | p1/p0*/r5 | | | \| | p1*/r4 | | |
| 4 | d | | | | \| | | | \| | | p2* | | \| | p2/p1* | | \| | \| | p2*/r6 | |
| 5 | | | | | \| | | | p3* | | \| | | \| | \| | p3/r7 | \| | \| | \| | p3*/r4 |
| 6 | | b | | | \| | | | \| | | \| | | • | \| | \| | \| | \| | \| | \| |
| 7 | | | a | | \| | | | \| | | \| | | | \| | \| | \| | \| | \| | \| |
| 8 | | d | | a | • | | | \| | | \| | | | | \| | • | p0/r1 | \| | \| | \| |
| 9 | | | d | | | | | \| | | \| | | | | \| | | | \| | \| | \| |
| 10 | | c | | | | | | p3 | | \| | | | | • | | | | \| | \| | p3/r4 |
| 11 | | | b | | | | | \| | | \| | | | | | | | \| | \| | \| |
| 12 | | | c | b | | | | \| | | \| | | | | | | p1/r4 | \| | \| |
| 13 | | | | c | | | | \| | | • | | | | | | | | p2/r6 | \| |
| 14 | | | | d | | | | • | | | | | | | | | | | | • |
| 15 | | | | | | | | | | | | | | | | | | | | |