Review Problem 51

- Show the constraint graph for this code, indicating the type of hazard for each edge.

1: LDUR X1, [X6, #8]
2: ADD X2, X1, X6
3: LDUR X3, [X7, #16]
4: SUB X4, X3, X8
5: STUR X5, [X9, #0]
6: CBZ X15, FOO
Hazard Minimization

**Hardware and software techniques for improving performance**

**Sw:** Loop unrolling

**Sw/Hw:** Register Renaming

**Hw/Sw:** Predicated Instructions

**Responsibility**
- Hardware
- Software (Compiler)
Loop-Level Parallelism

LLPex(char *src) {
    char *end = src + 1000;
    while(src<end) {
        *src = (*src)+3; src++;
    }
}

; x0 = src, x1 = end, x2 = *src
    ADDI x1, x0, #1000  ; end = src + 1000
    LOOP:
    LDRB  x2, [x0, #0]  ; set *src
    ADDI x8, x2, #3     ; compute (*src) + 3
    STURB x2, [x0, #10] ; store to *src
    ADDI x0, x8, #1     ; src++
    CMP x0, x1          ; compare src + end
    B.LT LOOP

6 instructions / iteration
Completely seq.
```c
char *end = src + 1000;
```
Loop Unrolling

Better loop structure?

```
LUPc (char *src)
    char *end = src + 1000;
    while (src != end)
        *src = (*src) + 3;
        *(src + 1) = (*(src + 1)) + 3;
        src += 2;

// x0 = src, x1 = end, x2 = src
ADDI x, x0, #1000
LOOP:
  LDURB x2, [x0, #0]; ADDI x2, x2, #3; STURB x2, [x0, #1]; // modify *src
  LDURB x2, [x0, #1]; ADDI x2, x2, #3; STURB x2, [x0, #1]; // modify *(src + 1)
  ADDI x0, x0, #2; // src += 2
  CMP x0, x1
  B, LT LOOP
```

9 instructions
---
2 iterations
4.5 instructions/array location, array locations
Compiler Register Renaming

Compiler reduces hazards by removing name dependences

Avoid using the same register

Previous slide in blue

LDURB X2
ADDI x2
STURB X2

LDURB X3
ADDI x3
STURB X3

WAR X0
ADDI X0

cmp d
b, lt
Hardware Register Renaming

CPU dynamically associates each register read with the most recent instruction issue that writes that register.

0: ADDI X0, X0, #4
1: LDUR X1, [X0, #0]
2: STUR X1, [X0, #100]
3: ADDI X0, X0, #4
4: LDUR X1, [X0, #0]
5: STUR X1, [X0, #100]
6: ADDI X0, X0, #4
7: LDUR X1, [X0, #0]
8: STUR X1, [X0, #100]
Hardware Register Renaming (cont.)
Review Problem 52

• Would loop unrolling & register renaming be useful for the following code? If so, what would the resulting code look like?

```c
while (i<400) {
    if (x[i]==CONST) counter++; /* Count number of CONSTs in array */
    i++;
}
```

```c
while (i<399)
    if(x[i]==CONST) counter++;
    if(x[i+1]==CONST) counter++;
    i+=2;

while (i<400)
    if(x[i]==CONST) counter++;
    i++;
```

ADDI  counter, counter, #1
LD    B.
CMP DONE
ADDI  DONE:
Control Hazards

```c
if (c == 0) { t = s; }
if (a[0] == 0)
    a[0] = b[0];
else
    a[0] = a[0] + 4;
```

Branches introduce hazards that limit ILP
   Branch prediction

   Conditional/Predicated instructions
Predicated Instructions

if (c == 0) { t = s; }

Normal:
CBNZ c, ENDF
ADDI t, s, #0

ENDIF:

W/Conditional move (instructions with internal if-like operation – no branches)
CMOVZ <dest>, <src>, <cond>  // move src to dest if cond == 0
CMOVNZ <dest>, <src>, <cond>  // move src to dest if cond != 0

Note: ARM actually uses CSEL, a Mux-like instruction instead. But, predication important enough we’ll pretend there’s a CMOV instruction…
Predicated Instructions (cont.)

\[
d = a - b;
\]

\[
\text{if } (d < 0) \quad d < 0
\]

\[
\text{sum } -= d;
\]

\[
\text{else}
\]

\[
\text{sum } += d;
\]

Predicated Instructions:

1: SUB D, A, B  \quad /\text{Sub + set flags}\n
2: SUB t1, sum, D

3: MOVADD t2, sum, D

4: LSR t3, D, #63  \quad /t3:1<0\text{ D mask}\n
5: CMOV2 sum, t2  \quad \text{if } t3

6: CMOV2 sum, t1  \quad \text{if } t3

\text{Normal:}

- SUBS D, A, B
- BGE ELSE
- SUB sum, sum, D
- B END

ELSE:

- ADD sum, sum, D

END:

\text{3-4 instr}  \quad \text{almost completely sequential}

\text{6 instr}
ARM CSEL Instruction: the MUX

CSEL <dest>, <src1>, <src2>, <cond>

<dest> = If <cond> then <src1> else <src2>
<cond>: EQ, NE, LT, etc.

d = a - b;
if (d<0)
    sum -= d;
else
    sum += d;

SUBS d, a, b  //sets flags
SUB t1, sum, d  //d<0 result
ADD t2, sum, d  //d>0 result
CSEL sum, t1, t2, LT