Lab 3: Programming on Modern Processors

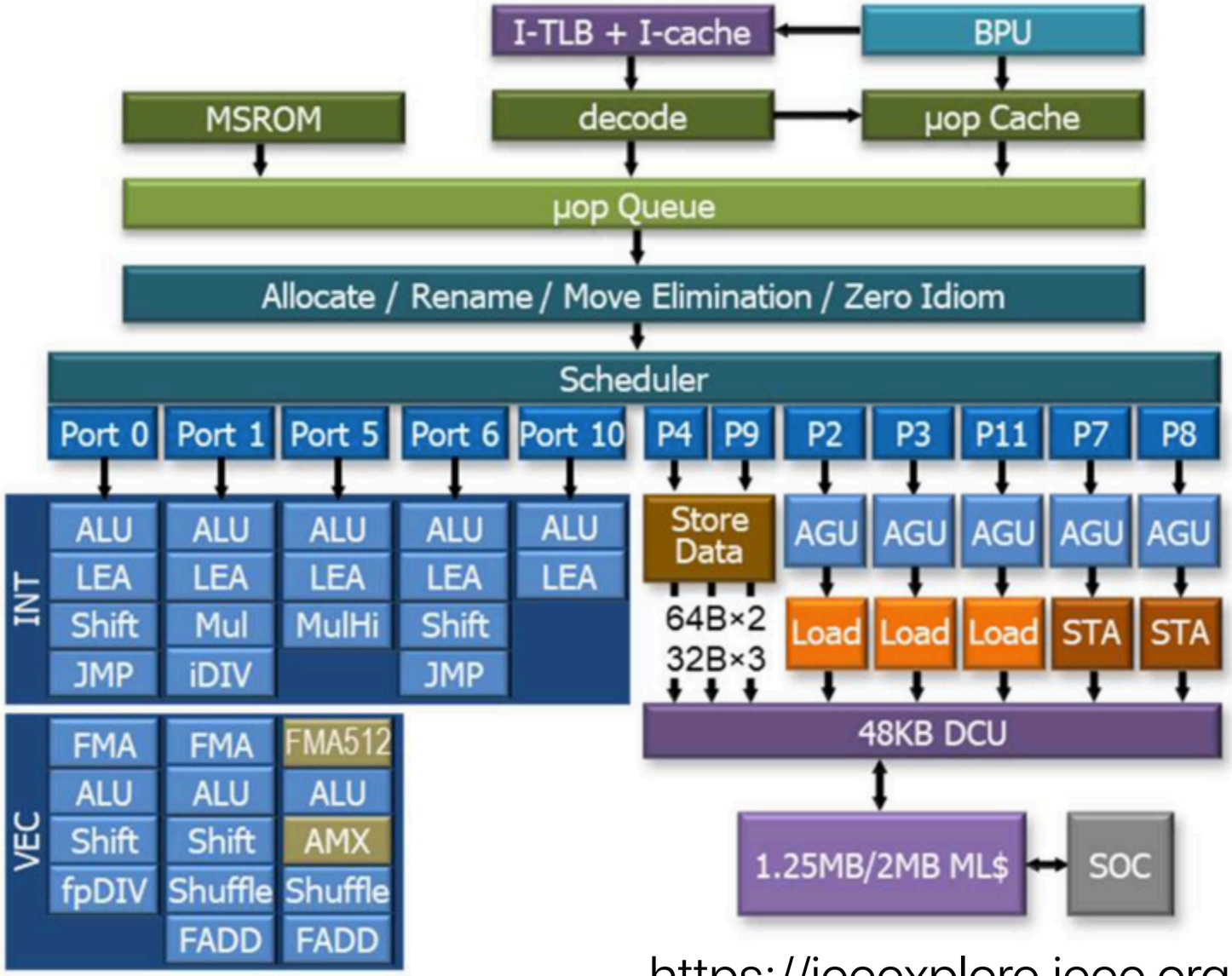
Hung-Wei Tseng

Know your Pokémon!

```
[2]: ! cse142 job run 'lscpu'

Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              2
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           151
Model name:                      12th Gen Intel(R) Core(TM) i3-12100F
Stepping:                        5
CPU MHz:                         4000.000
CPU max MHz:                     5500.0000
CPU min MHz:                     800.0000
BogoMIPS:                        6604.80
Virtualization:                  VT-x
L1d cache:                       192 KiB
L1i cache:                       128 KiB
L2 cache:                        5 MiB
L3 cache:                        12 MiB
NUMA node0 CPU(s):               0-7
Vulnerability Itlb multihit:     Not affected
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Not affected
Vulnerability Meltdown:          Not affected
Vulnerability Mmio stale data:   Not affected
Vulnerability Retbleed:          Not affected
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Enhanced IBRS, IBPB conditional, RSB filling, PBRSB-eIBRS SW sequence
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Not affected
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts a
cpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl x
topology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fma cx
16 xtpr pdcm sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch
cpuid_fault ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_adjust bmi1 a
vx2 smep bmi2 erms invpcid rdseed adx smap clflushopt clwb intel_pt sha_ni xsaveopt xsavec xgetbv1 xsaves split_lock_d
etect avx_vnni dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp hwp_pkg_req umip pku ospke waitpkg gfni v
aes vpclmulqdq rdpid movdiri movdir64b fsrm md_clear serialize arch_lbr flush_l1d arch_capabilities
```

The Intel "Alder Lake" Architecture



https://ieeexplore.ieee.org/document/9747991

How can we use the processor efficiently?

Outline

- Compiler optimizations
- Code that can exploit instruction level parallelism

Let’s start with a code snippet

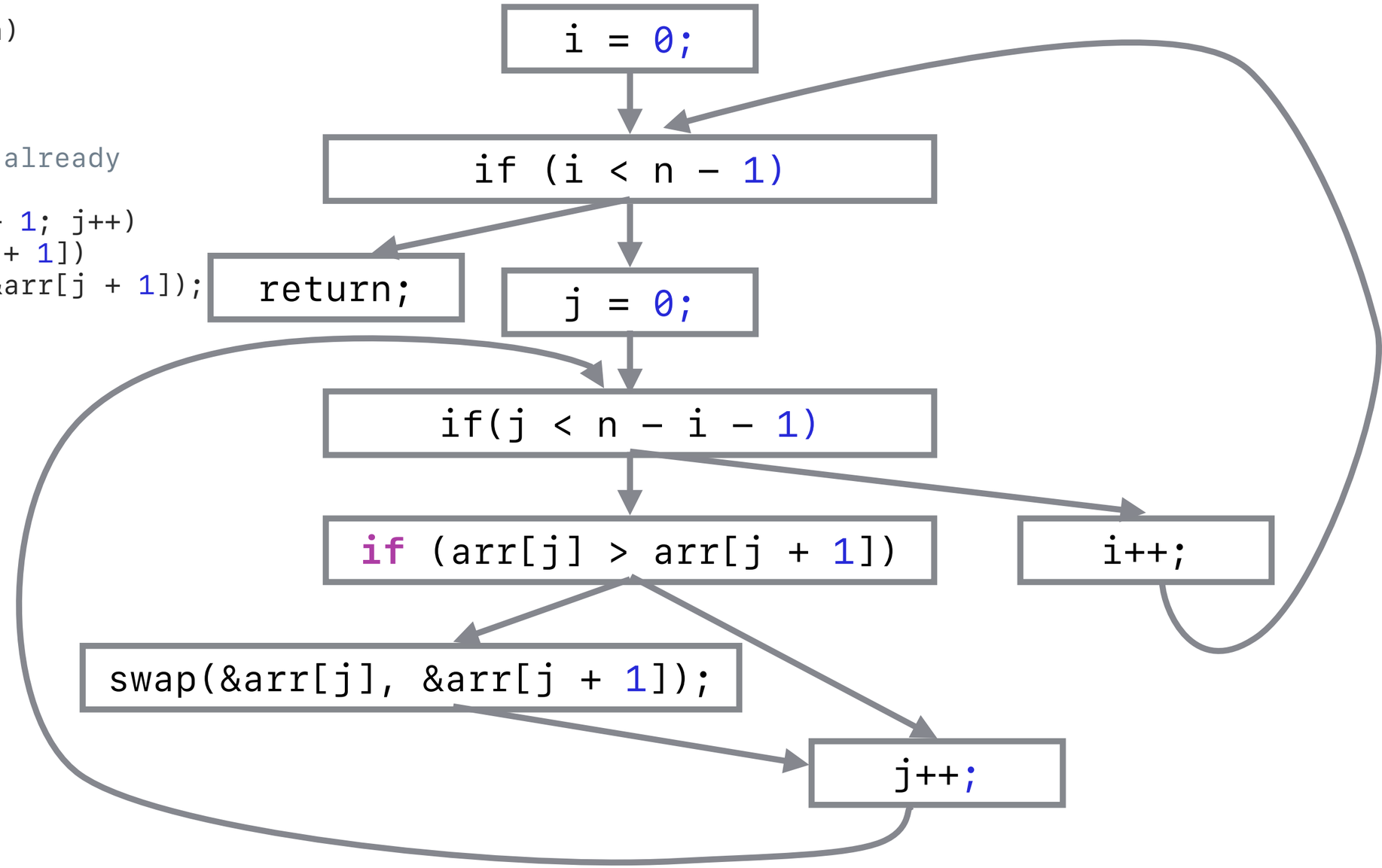
```
// A function to implement bubble sort
void bubbleSort(int *arr, int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)
        // Last i elements are already
        // in place
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(&arr[j], &arr[j + 1]);
}
```

Control flow graph

- A graph shows all possible route of executing a piece code
- A “directed” graph consists of basic blocks
- A basic block is a sequence of instructions that will always execute together
  - A sequence of instructions until we reach a “branch”

CFG example

```
void bubbleSort(int *arr, int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)
        // Last i elements are already
        // in place
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(&arr[j], &arr[j + 1]);
}
```



Translate from C to Assembly

- gcc: gcc [options] [src_file]
  - compile to binary
    - gcc -o foo foo.c
  - compile to assembly (assembly in foo.s)
    - gcc -S foo.c
  - **compile with debugging message**
    - **gcc -g -S foo.c**
  - optimization
    - gcc -O*n* -S foo.c
      - n from 0 to 3 (0 is no optimization)

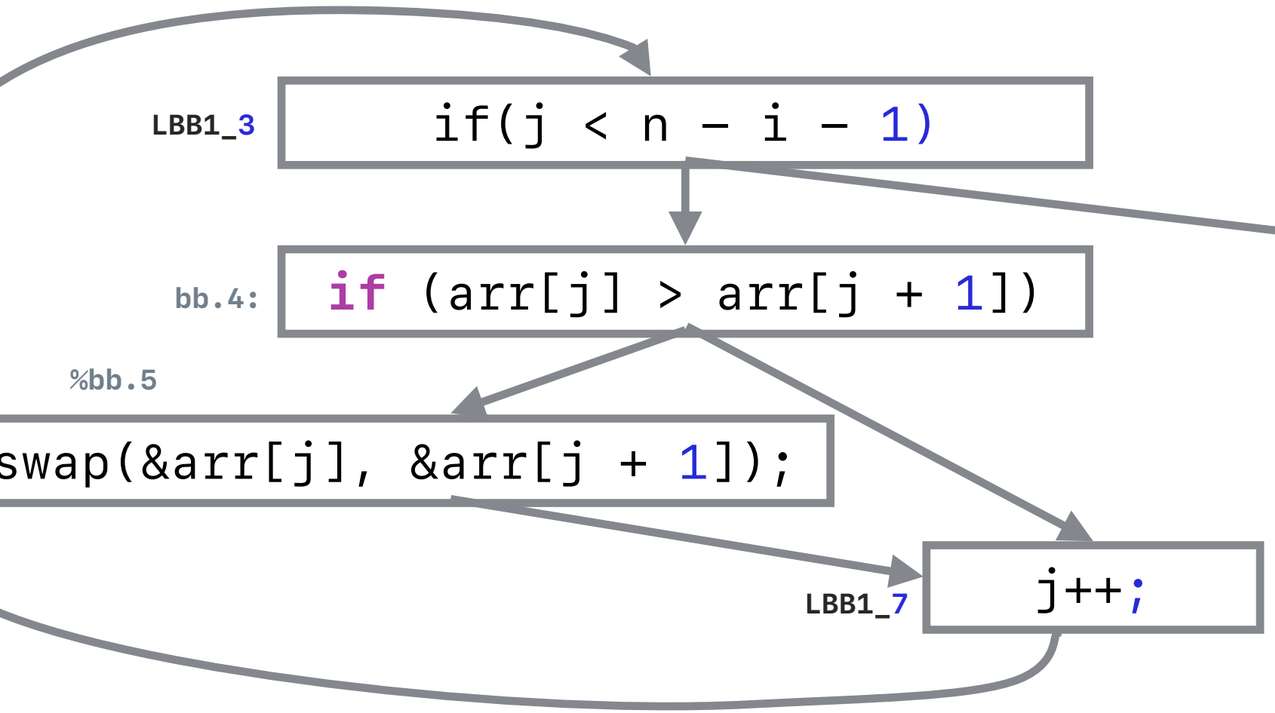
```
    .globl _bubbleSort                      ## -- Begin function
bubbleSort
    .p2align  4, 0x90
_bubbleSort:                            ## @bubbleSort
    .cfi_startproc
## %bb.0:
    pushq  %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset %rbp, -16
    movq   %rsp, %rbp
    .cfi_def_cfa_register %rbp
    subq   $32, %rsp
    movq   %rdi, -8(%rbp)
    movl   %esi, -12(%rbp)
    movl   $0, -16(%rbp)
LBB1_1:                                 ## =>This Loop Header:
Depth=1

                                        ##     Child Loop BB1_3
Depth 2
    movl   -16(%rbp), %eax
    movl   -12(%rbp), %ecx
    subl   $1, %ecx
    cmpl   %ecx, %eax
    jge LBB1_10
## %bb.2:                               ##   in Loop:
Header=BB1_1 Depth=1
    movl   $0, -20(%rbp)
LBB1_3:                                 ##   Parent Loop BB1_1
Depth=1

                                        ## =>  This Inner Loop
Header: Depth=2
    movl   -20(%rbp), %eax
    movl   -12(%rbp), %ecx
    subl   -16(%rbp), %ecx
    subl   $1, %ecx
    cmpl   %ecx, %eax
    jge LBB1_8
## %bb.4:                               ##   in Loop:
Header=BB1_3 Depth=2
    movq   -8(%rbp), %rax
    movslq -20(%rbp), %rcx
    movl   (%rax,%rcx,4), %eax
    movq   -8(%rbp), %rcx
    movl   -20(%rbp), %edx
    addl   $1, %edx
    movslq %edx, %rdx
    cmpl   (%rcx,%rdx,4), %eax
    jle LBB1_6
## %bb.5:                               ##   in Loop:
Header=BB1_3 Depth=2
    movq   -8(%rbp), %rdi
    movslq -20(%rbp), %rax
    shlq   $2, %rax
    addq   %rax, %rdi
    movq   -8(%rbp), %rsi
    movl   -20(%rbp), %eax
    addl   $1, %eax
    cltq
    shlq   $2, %rax
    addq   %rax, %rsi
    callq  _swap
LBB1_6:                                 ##   in Loop:
Header=BB1_3 Depth=2
    jmp LBB1_7
LBB1_7:                                 ##   in Loop:
Header=BB1_3 Depth=2
    movl   -20(%rbp), %eax
    addl   $1, %eax
    movl   %eax, -20(%rbp)
    jmp LBB1_3
LBB1_8:                                 ##   in Loop:
Header=BB1_1 Depth=1
    jmp LBB1_9
LBB1_9:                                 ##   in Loop:
Header=BB1_1 Depth=1
    movl   -16(%rbp), %eax
    addl   $1, %eax
    movl   %eax, -16(%rbp)
    jmp LBB1_1
LBB1_10:
    addq   $32, %rsp
    popq   %rbp
    retq
    .cfi_endproc
```

Take a look of the inner loop

```
for (j = 0; j < n - i - 1; j++)
    if (arr[j] > arr[j + 1])
        swap(&arr[j], &arr[j + 1]);
```




```
LBB1_3:
## =>  This Inner Loop Header: Depth=2
    movl   -20(%rbp), %eax      ## %eax = j
    movl   -12(%rbp), %ecx      ## %ecx = n
    subl   -16(%rbp), %ecx      ## %ecx = %ecx - i
    subl   $1, %ecx             ## %ecx = %ecx - 1
    cmpl   %ecx, %eax           ## if (result >= 0), go to LBB1_8:
    jge LBB1_8                  ## result = %ecx - %eax
## %bb.4:                                ##   in Loop: Header=BB1_3 Depth=2
    movq   -8(%rbp), %rax
    movslq -20(%rbp), %rcx
    movl   (%rax,%rcx,4), %eax
    movq   -8(%rbp), %rcx
    movl   -20(%rbp), %edx
    addl   $1, %edx
    movslq %edx, %rdx
    cmpl   (%rcx,%rdx,4), %eax
    jle LBB1_6
## %bb.5:                                ##   in Loop: Header=BB1_3 Depth=2
    movq   -8(%rbp), %rdi
    movslq -20(%rbp), %rax
    shlq   $2, %rax
    addq   %rax, %rdi
    movq   -8(%rbp), %rsi
    movl   -20(%rbp), %eax
    addl   $1, %eax
    cltq
    shlq   $2, %rax
    addq   %rax, %rsi
    callq  _swap
LBB1_6:                                   ##   in Loop: Header=BB1_3 Depth=2
    jmp LBB1_7
LBB1_7:                                   ##   in Loop: Header=BB1_3 Depth=2
    movl   -20(%rbp), %eax
    addl   $1, %eax
    movl   %eax, -20(%rbp)
    jmp LBB1_3
```

Now, with -O1

```
LBB1_7:
    addl   $1, %r13d
    addl   $-1, %r12d
    cmpl   %r15d, %r13d
    je  LBB1_8
LBB1_2:

    movl   %r12d, %r12d
    movl   %r13d, %eax
    notl   %eax
    addl   -56(%rbp), %eax
    testl  %eax, %eax
    jle LBB1_7
## %bb.3:
    movq   %r12, %r14
    movq   -48(%rbp), %rbx
    jmp LBB1_4
    .p2align   4, 0x90
LBB1_6:
    addq   $4, %rbx
    addq   $-1, %r14
    je  LBB1_7
LBB1_4:

Depth=2
    movl   -4(%rbx), %eax
    cmpl   (%rbx), %eax
    jle LBB1_6
## %bb.5:
    leaq   -4(%rbx), %rdi
    movq   %rbx, %rsi
    callq  _swap
    jmp LBB1_6
```

```
##   in Loop: Header=BB1_2 Depth=1
## %r13d= i
## %r12d= n-1


## =>This Loop Header: Depth=1
##     Child Loop BB1_4 Depth 2

## %ecx = i
## not —> 2's compliment of i ( -i + 1)
## -56(%rbp) = n -> %eax =  n -i

## if (result >= 0), go to LBB1_7:
##   in Loop: Header=BB1_2 Depth=1

## 8-byte Reload


##   in Loop: Header=BB1_4 Depth=2



##   Parent Loop BB1_2 Depth=1
## =>  This Inner Loop Header:



##   in Loop: Header=BB1_4 Depth=2
```

```
LBB1_3:
## =>  This Inner Loop Header: Depth=2
    movl   -20(%rbp), %eax
    movl   -12(%rbp), %ecx
    subl   -16(%rbp), %ecx
    subl   $1, %ecx
    cmpl   %ecx, %eax
    jge LBB1_8
## %bb.4:                                ##   in Loop: Header=BB1_3 Depth=2
    movq   -8(%rbp), %rax
    movslq -20(%rbp), %rcx
    movl   (%rax,%rcx,4), %eax
    movq   -8(%rbp), %rcx
    movl   -20(%rbp), %edx
    addl   $1, %edx
    movslq %edx, %rdx
    cmpl   (%rcx,%rdx,4), %eax
    jle LBB1_6
## %bb.5:                                ##   in Loop: Header=BB1_3 Depth=2
    movq   -8(%rbp), %rdi
    movslq -20(%rbp), %rax
    shlq   $2, %rax
    addq   %rax, %rdi
    movq   -8(%rbp), %rsi
    movl   -20(%rbp), %eax
    addl   $1, %eax
    cltq
    shlq   $2, %rax
    addq   %rax, %rsi
    callq  _swap
LBB1_6:                                   ##   in Loop: Header=BB1_3 Depth=2
    jmp LBB1_7
LBB1_7:                                   ##   in Loop: Header=BB1_3 Depth=2
    movl   -20(%rbp), %eax
    addl   $1, %eax
    movl   %eax, -20(%rbp)
    jmp LBB1_3
```

Optimization with -O1

- Some variables are now in registers
- Leading to fewer memory accesses
- Load Effective Address (lea)
  - Reduce memory address calculations
- Strength Reduction — use a simpler operation
  - notl v.s. two subs

Now, with -O2

```
LBB1_16:                                ##   in Loop: Header=BB1_11 Depth=2
    movl   %edx, 4(%rdi,%rbx,4)
    movl   %eax, 8(%rdi,%rbx,4)
LBB1_17:                                ##   in Loop: Header=BB1_11 Depth=2
    movq   %rcx, %rbx
    cmpq   %rcx, %r14
    je  LBB1_5
LBB1_11:                                ##   Parent Loop BB1_2 Depth=1
                                        ## =>  This Inner Loop Header: Depth=2
    movl   4(%rdi,%rbx,4), %ecx
    cmpl   %ecx, %eax
    jle LBB1_12
## %bb.13:                              ##   in Loop: Header=BB1_11 Depth=2
    movl   %ecx, (%rdi,%rbx,4)
    movl   %eax, 4(%rdi,%rbx,4)
    jmp LBB1_14
    .p2align   4, 0x90
LBB1_12:                                ##   in Loop: Header=BB1_11 Depth=2
    movl   %ecx, %eax
LBB1_14:                                ##   in Loop: Header=BB1_11 Depth=2
    leaq   2(%rbx), %rcx
    movl   8(%rdi,%rbx,4), %edx
    cmpl   %edx, %eax
    jg  LBB1_16
## %bb.15:                              ##   in Loop: Header=BB1_11 Depth=2
    movl   %edx, %eax
    jmp LBB1_17
```

What's this?

```
LBB1_7:                                 ##   in Loop: Header=BB1_2 Depth
    addl   $1, %r13d
    addl   $-1, %r12d
    cmpl   %r15d, %r13d
    je  LBB1_8
LBB1_2:                                 ## =>This Loop Header: Depth=1
                                        ##     Child Loop BB1_4 Depth 2
    movl   %r12d, %r12d
    movl   %r13d, %eax
    notl   %eax
    addl   -56(%rbp), %eax              ## 4-byte Folded Reload
    testl  %eax, %eax
    jle LBB1_7
## %bb.3:                               ##   in Loop: Header=BB1_2 Depth
    movq   %r12, %r14
    movq   -48(%rbp), %rbx              ## 8-byte Reload
    jmp LBB1_4
    .p2align   4, 0x90
LBB1_6:                                  ##   in Loop: Header=BB1_4 Dep
    addq   $4, %rbx
    addq   $-1, %r14
    je  LBB1_7
LBB1_4:                                  ##   Parent Loop BB1_2 Depth=1
                                         ## =>  This Inner Loop Header:
Depth=2
    movl   -4(%rbx), %eax
    cmpl   (%rbx), %eax
    jle LBB1_6
## %bb.5:                                ##   in Loop: Header=BB1_4 Dep
    leaq   -4(%rbx), %rdi
    movq   %rbx, %rsi
    callq  _swap
    jmp LBB1_6
```

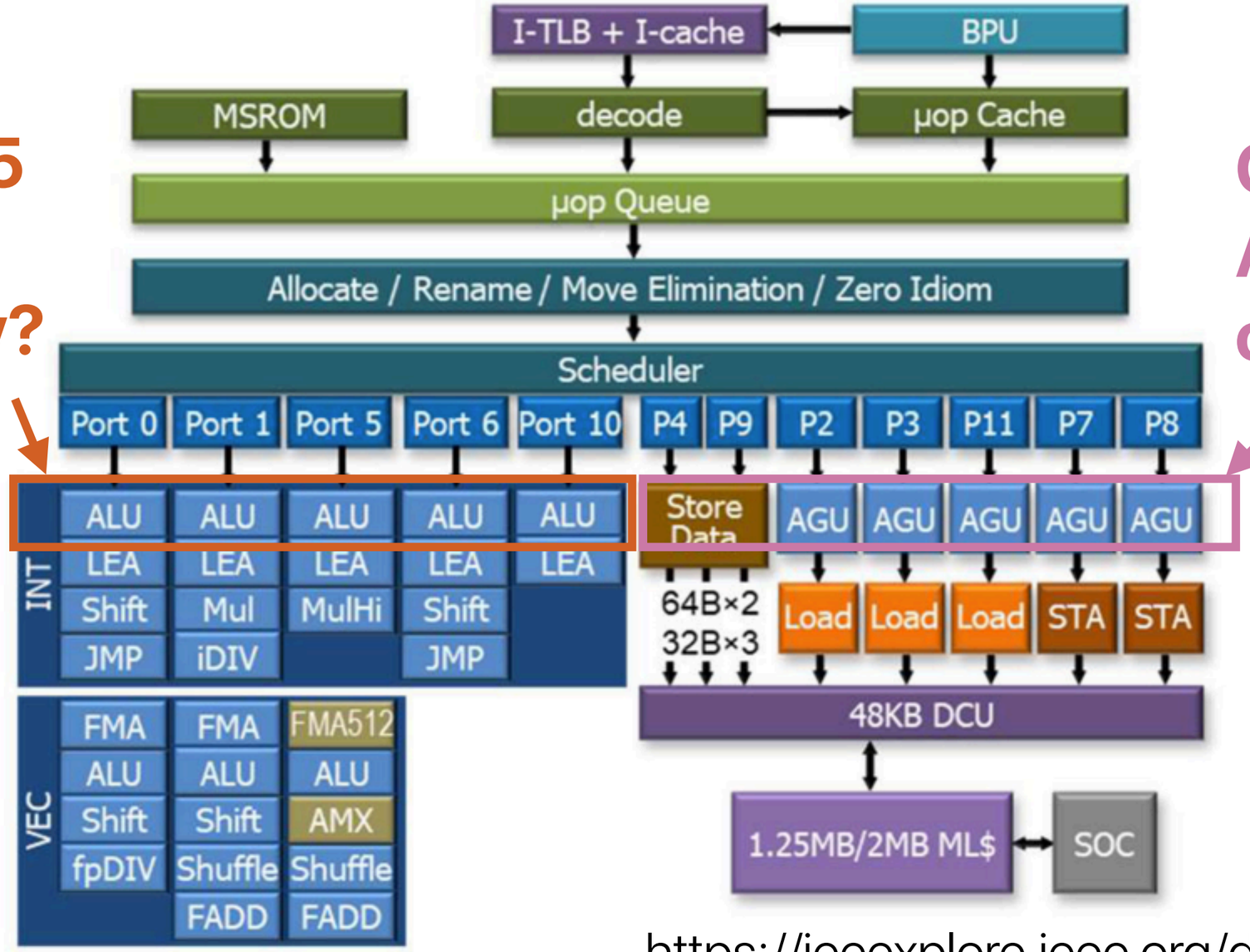
-O2

- Loop invariant code motion
- Constant propagation
- Function Inlining
- More you can find when you do your lab 2!

The Intel "Alder Lake" Architecture

Can we use 5 ALUs concurrently?

Can we use 5 AGUs concurrently?



https://ieeexplore.ieee.org/document/9747991

What are the characteristics of code with good ILP?

```
uint64_t* wide_1(uint64_t threads, uint64_t *
data, uint64_t size, uint64_t arg1, uint64_t
arg2, uint64_t arg3) {
    register uint64_t i =0;
    for(i = 0; i < size; i++) {
        i = i+1;
        i = i+1;
        i = i+1;
        i = i+1;
        i = i+1;
    }
    data[0] = i;
    return data;
}
```

```
uint64_t* wide_2(uint64_t threads,
uint64_t * data, uint64_t size, uint64_t
arg1, uint64_t arg2, uint64_t arg3) {
    register uint64_t a = 4;
    register uint64_t b = 4;
    register uint64_t c = 4;
    register uint64_t d= 4;
    register uint64_t e =4 ;
    register uint64_t f= size;

    for(register uint64_t i = 0; i <
size; i++) {
        i = i+1;  a = a+1;
        i = i+1;  a = a+1;
        i = i+1;  a = a+1;
        i = i+1;  a = a+1;
        i = i+1;  a = a+1;
    }
    data[0] = a + b + c +d + e + f;
    return data;
}
```

```
uint64_t* wide_5(uint64_t threads, uint64_t * data,
uint64_t size, uint64_t arg1, uint64_t arg2, uint64_t
arg3) {
    register uint64_t a = 4;
    register uint64_t b = 4;
    register uint64_t c = 4;
    register uint64_t d= 4;
    register uint64_t e =4 ;
    register uint64_t f= size;

    for(register uint64_t i = 0; i < size; i++) {
        i = i+1;  a = a+1; b = b+1;  d = d+1; e = e+1;
        i = i+1;  a = a+1; b = b+1;  d = d+1; e = e+1;
        i = i+1;  a = a+1; b = b+1;  d = d+1; e = e+1;
        i = i+1;  a = a+1; b = b+1;  d = d+1; e = e+1;
        i = i+1;  a = a+1; b = b+1;  d = d+1; e = e+1;
    }
    data[0] = a + b + c +d + e + f;
    return data;
}
```

Branch is slow!

Can you eliminate all branches here?




```
do {
     // V: taken if false
     if((pid[i] & 0x7) < 3)
           team[i]=YELLOW;
     else
     {
          // X: taken if false
          if((pid[i] & 0x7) == 3 || (pid[i] & 0x7) == 4)
            team[i]=BLUE;
          else
          {
            // Y: taken if false
              if((pid[i] & 0x7) == 5 || (pid[i] & 0x7) == 6)
                team[i]=RED;
              else
                team[i]=ROCKET;
          }
     }
// Z: i < total_number_of_students means taken
} while(++i<total_number_of_students);
```

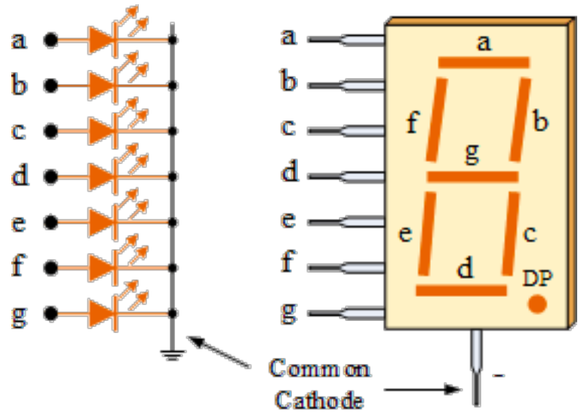
Ideas?

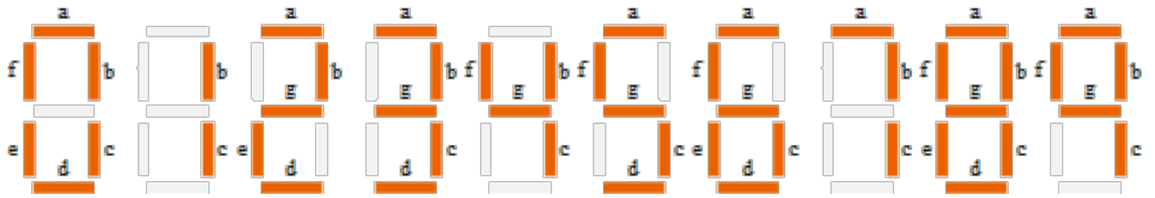
Programming assignment

7-segment display

- You need to transform binaries to “signals”

Decimal	a	b	c	d	e	f	g	Signal
0	x	x	x	x	x	x		7E
1		x	x					30
2	x	x		x	x		x	6D
3	x	x	x	x			x	79
4		x	x			x	x	33
5	x		x	x		x	x	5B
6	x		x	x	x	x	x	5F
7	x	x	x					70
8	x	x	x	x	x	x	x	7F
9	x	x	x			x	x	73
-							x	1





What you're given...

- Data dependencies?
- Are branch necessary?

```
void __attribute__((optimize("O4"))) seven_segConversion(char *r, int32_t n)
{
    // Base Case
    r[0]=0x0;
    if (n == 0) {
        return;
    }
    if (n < 0) {
        r[0]=0x1;
        n = (-1)*n;
    }

    // To store the reverse of n

    // Reversing the digits
    int pos=10;
    while (n > 0) {
        switch (n % 10)
        {
            case 1:
                r[pos]= 0x30;
                break;
            case 2:
                r[pos]= 0x6D;
                break;
            case 3:
                r[pos]= 0x79;
                break;
            case 4:
                r[pos]= 0x33;
                break;
            case 5:
                r[pos]= 0x5B;
                break;
            case 6:
                r[pos]= 0x5F;
                break;
            case 7:
                r[pos]= 0x70;
                break;
            case 8:
                r[pos]= 0x7F;
                break;
```

Computer
Science &
Engineering

142L

つづく