Contemporary Architecture

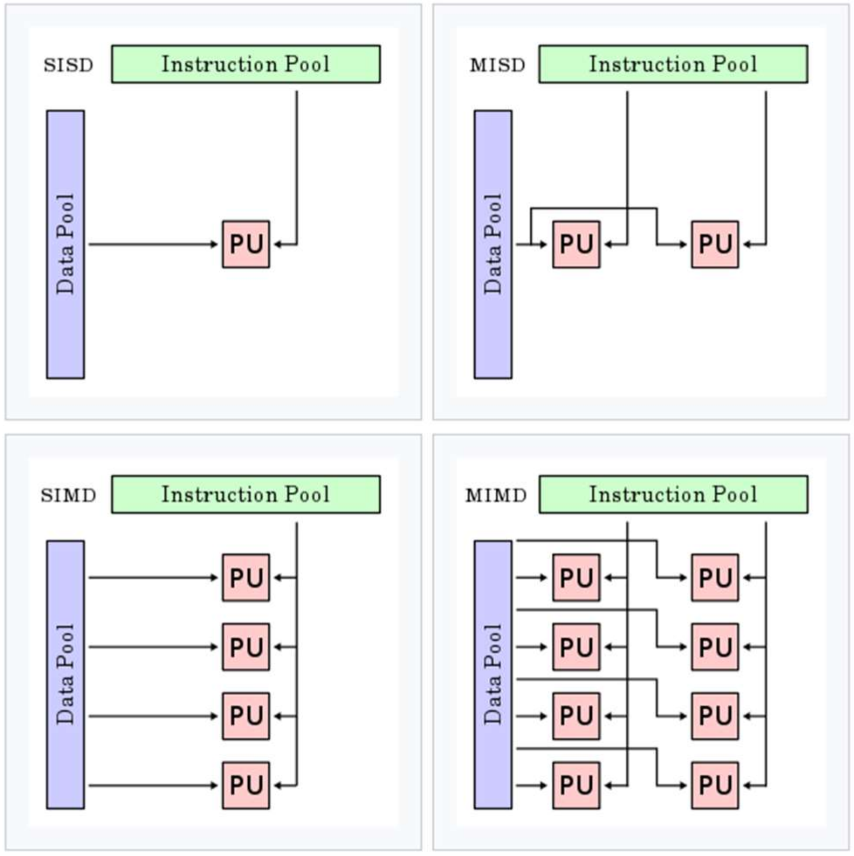
Chapter 7

Contemporary Architecture

- ★ Flynn's Taxonomy
  - ○ SISD, SIMD, MISD, MIMD
- ★ Instruction-Level Parallelism
  - ○ Vector Processor, Vector Unit
  - ○ Pipelining Processor, SuperPipeline
  - ○ SuperScalar (Out-of-order execution, Register Renaming)
  - ○ Very Long Instruction Word (VLIW)
- ★ SuperSymmetry
- ★ Dynamic vs. Static Scheduling
- ★ Compiler & ILP
  - ○ Loop Unroll, Software Pipelining, LLVM
- ★ Thread-Level Parallelism

Flynn’s Taxonomy

- ★ by Michael Flynn (1934-) in 1966
- ★ Partition by Instruction Stream and Data Stream
  - ○ Single Instruction stream,
    Single Data stream (SISD) - **uniprocessor**
  - ○ Single Instruction stream,
    Multiple Data stream (SIMD) - **vector processor**
  - ○ Multiple Instruction stream,
    Single Data stream (MISD) - **Fault-Tolerant Computing** (The course is available as approved from A. Arthit.)
  - ○ Multiple Instruction stream,
    Multiple Data stream (MISD) - **Multicore, Distributed Systems**



Pictures from wikipedia

Instruction-Level Parallelism

- ★ ILP - a measure of how many of the instructions in a computer program can be executed simultaneously
- ★ Several micro-architectural techniques involve:
  - ○ Vectorization
  - ○ Speculative Execution - allow (parts of) instructions to be executed before needed.
    - ■ Branch Prediction - avoid stalling from control dependencies(speculative execution)
  - ○ Instruction Pipelining and Super Pipeline
  - ○ SuperScalar
    - ■ Out-of-order execution
    - ■ Register Renaming
  - ○ VLIW

Vector Processor, Vector Unit

- ★ A variation of SIMD
- ★ Known products
  - ○ Apple/Motorola/IBM Altivec
  - ○ Intel MMX, SSE[2,3,4], AVX[2,512]
  - ○ GPU - CUDA, OpenCL
- ★ Apply single operation to vectors (array of data)
- ★ We will revisit this on GPGPU topics.

```
// uniprocessor
int A[8];
int B[8];
int C[8];

for (int i=0;i<8;i++)
{
    C[i]=A[i]+B[i];
}
```

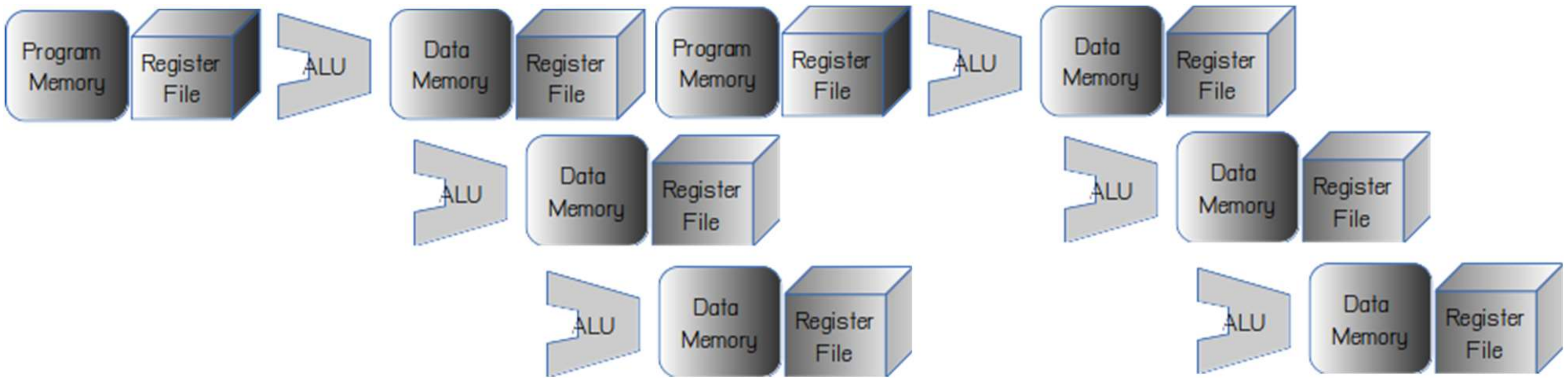
```
// vectorize code
// equivalent in C
int A[8];
int B[8];
int C[8];

C=A+B;
```

```
; Optimize of -O3 on GCC 6.3
; vectorize with Intel AVX
….
movdqa    16+B(%rip), %xmm0
paddd    16+A(%rip), %xmm0
movaps    %xmm0, 16+C(%rip)
….
```

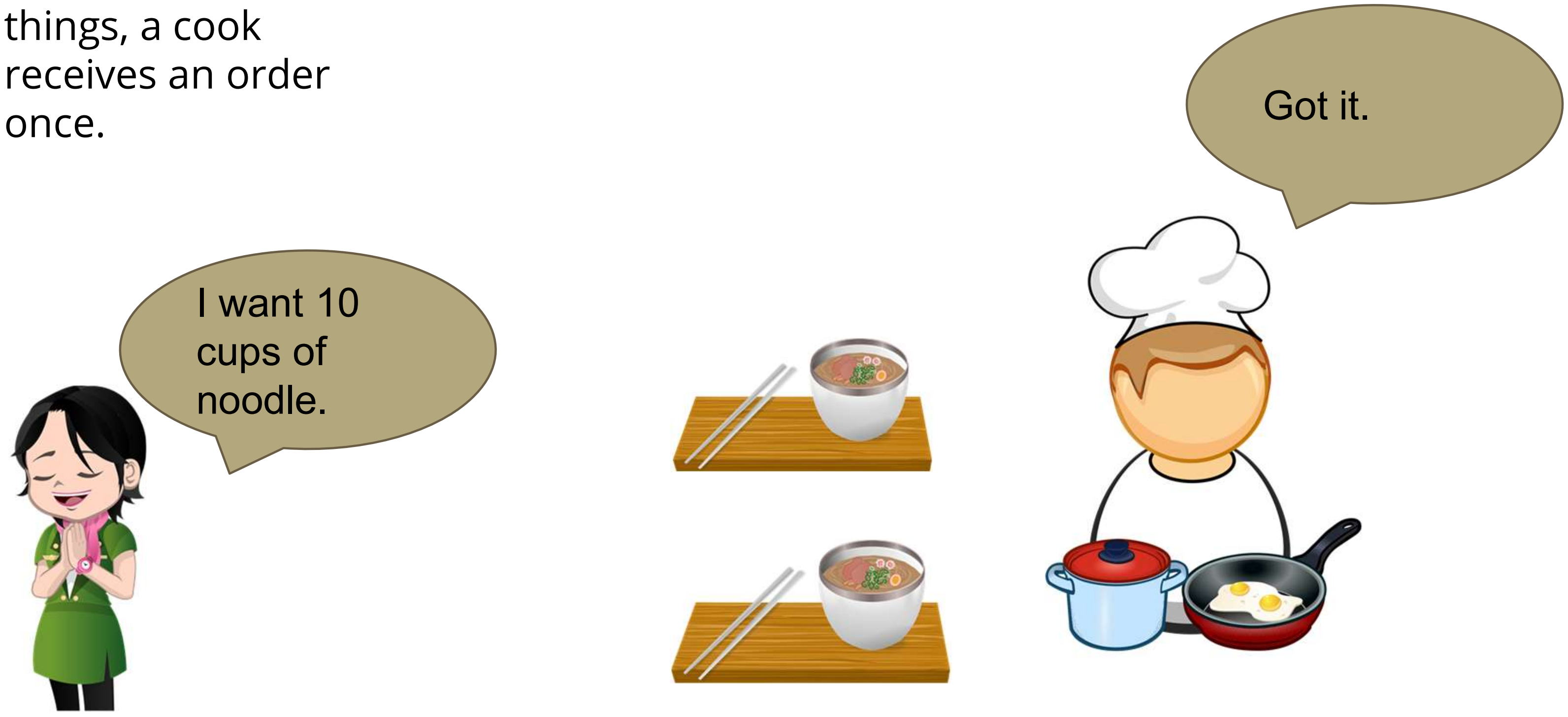
Vector Processor (ctd.)

★ Some vector processors will expand loops internally.



Vector processor in real world

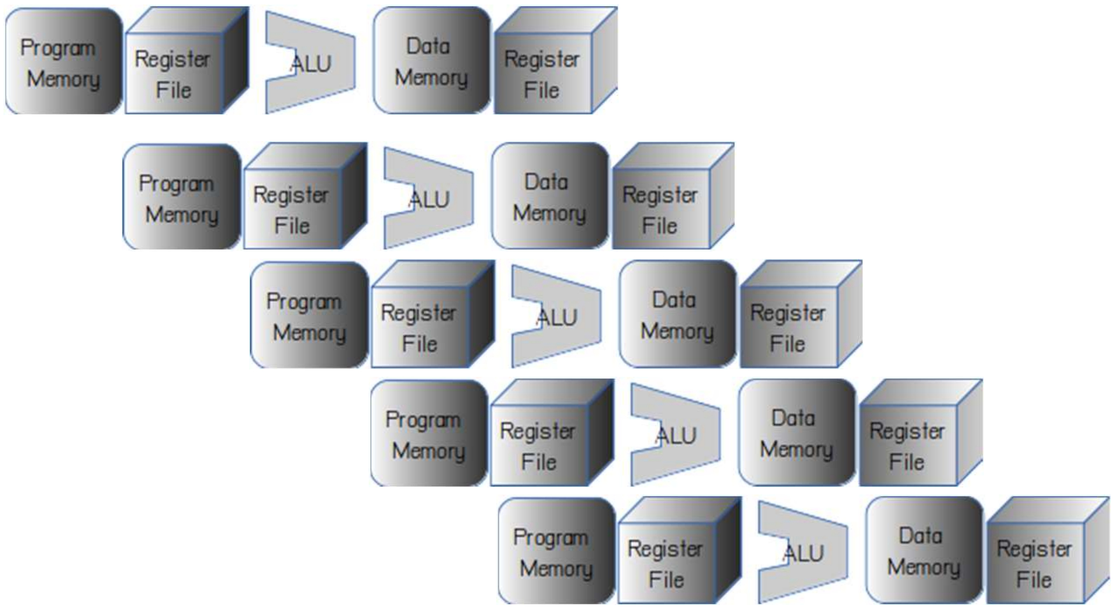
★ Ordering several dishes of the same things, a cook receives an order once.



Pipelining Processor (revisit)

★ A pipeline with 5 states has 5 instructions in flight.
★ For k state pipelines, ILP = k.

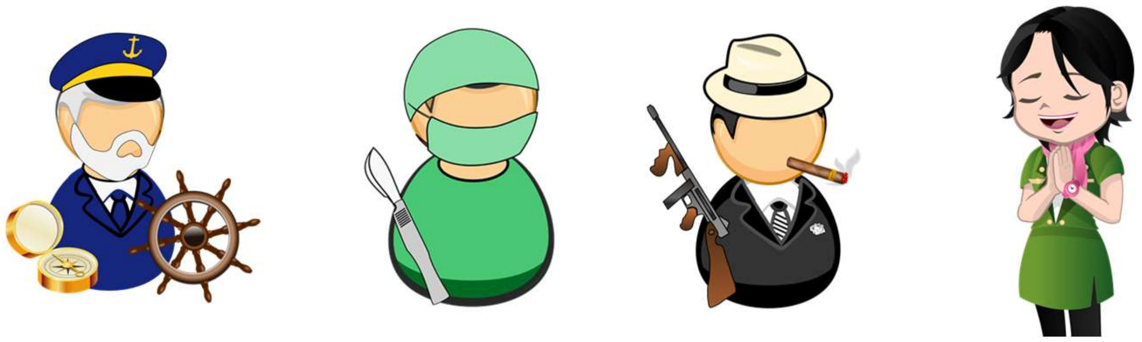
★ CPU Time for n instructions
T(n) = (k + (n-1)) Tc



Pipeline in real world

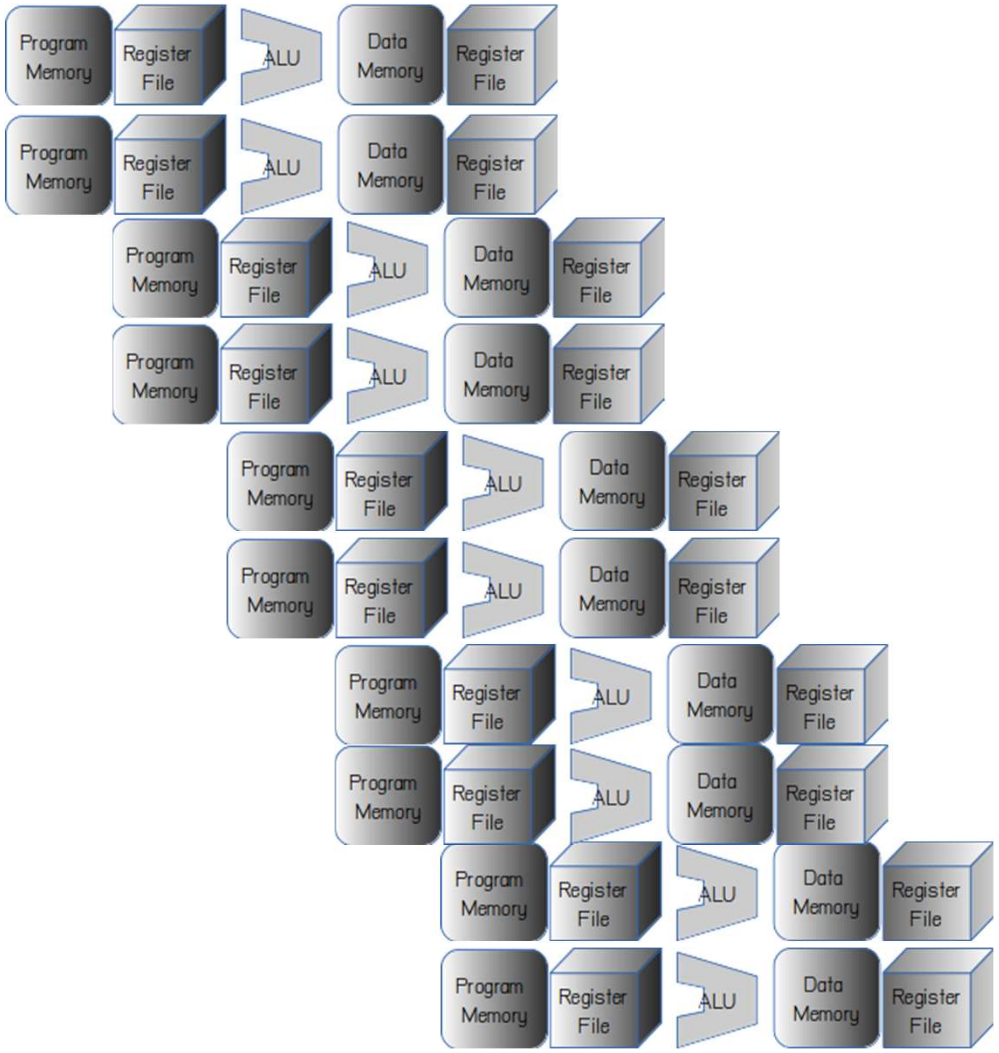
- ★ One line for buying food.
- ★ 5 steps

Chicken Nuggets



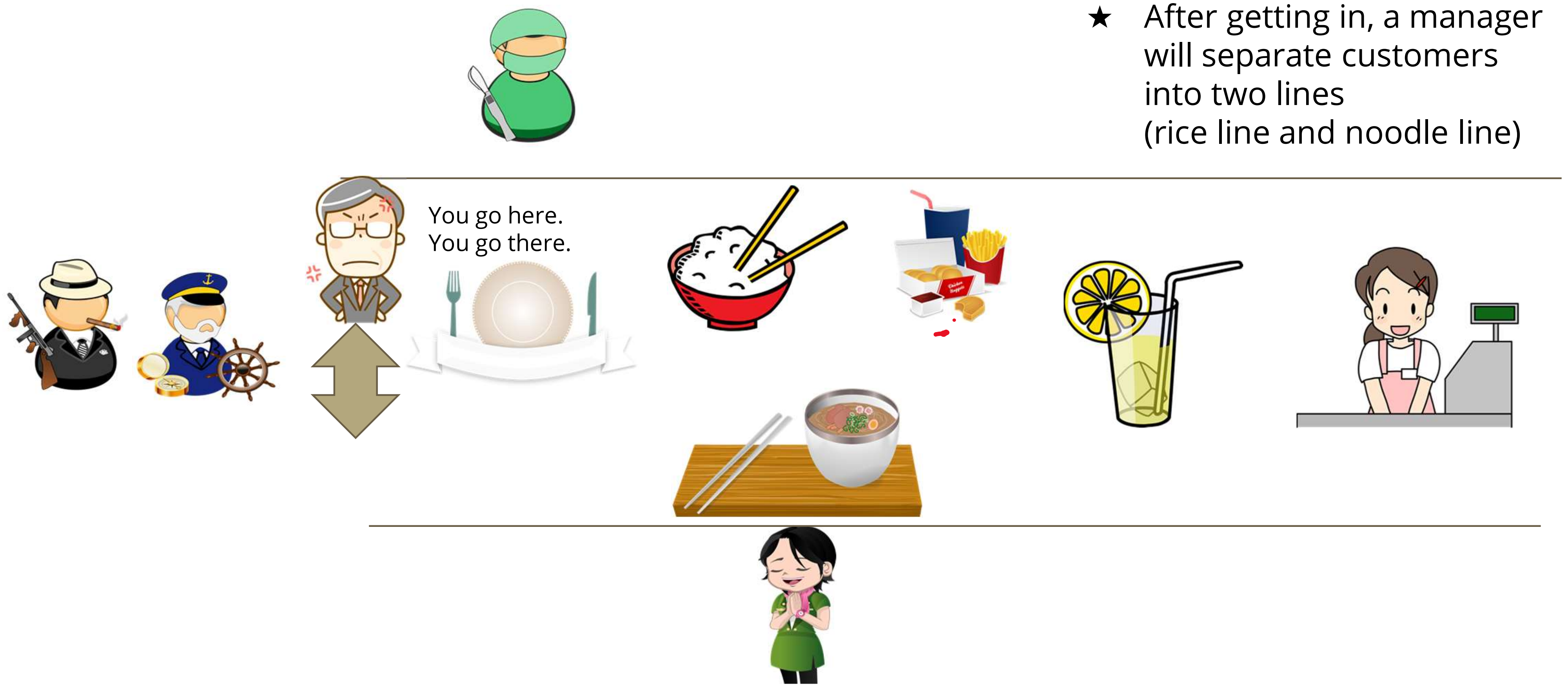
SuperScalar

- ★ SuperScalar with degree 2 and 5 states of pipeline has 10 instructions in flight.
- ★ For k state pipelines in superscalar x degrees, ILP = kx.
- ★ CPU Time for n instructions T(n) = (k + (n-x)/x) Tc



SuperScalar in real world

- ★ One line for buying food.
- ★ After getting in, a manager will separate customers into two lines (rice line and noodle line)

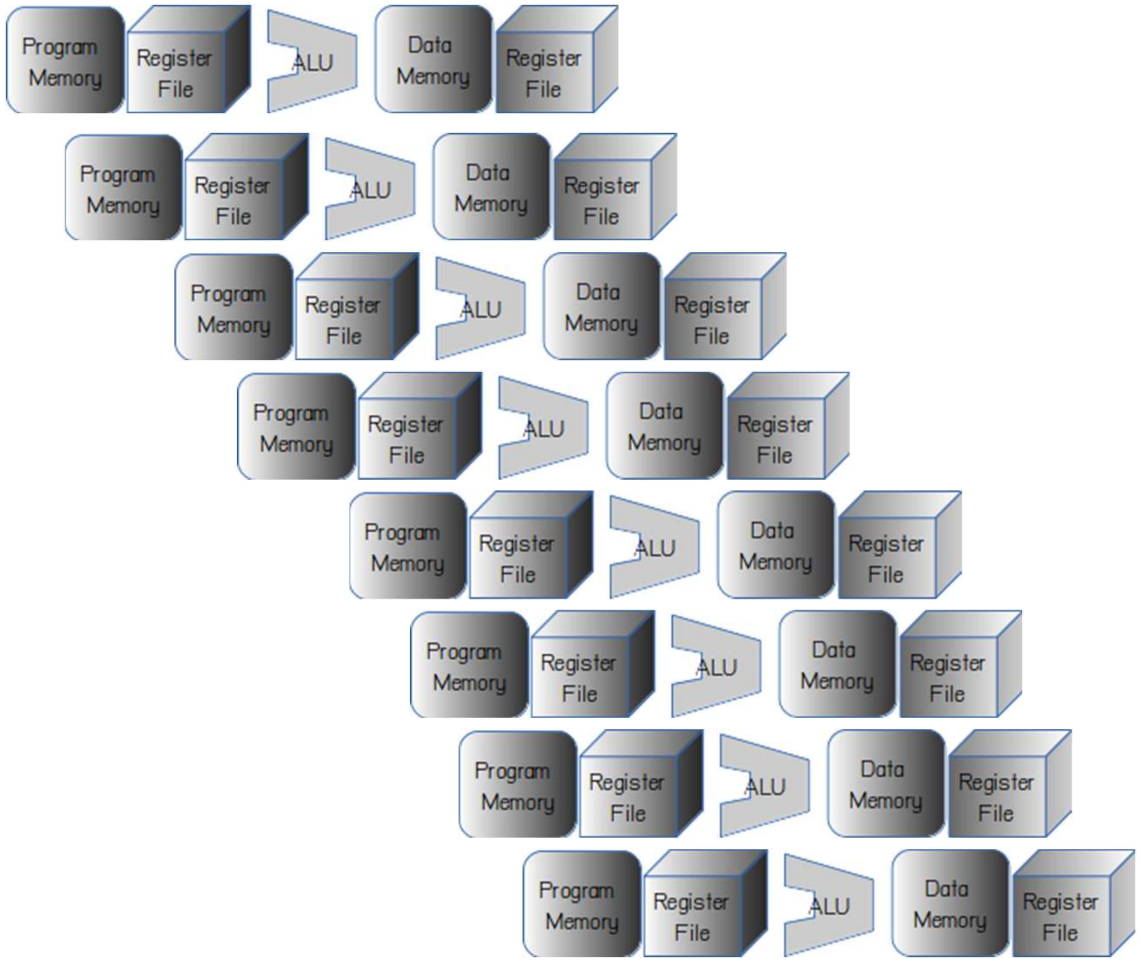


SuperScalar (ctd.)

★ With super scalar, we may have **out-of-order execution**.
★ A person that comes after (but order noodle) may finish earlier than a person that comes before (but order rice).
★ This may result in potential **WAR** and **WAW**.
★ To solve this -- **in-order completion** ?

SuperPipeline

- ★ Use minor cycles.
- ★ In this example, 2 minor cycles is equivalent to 1 major cycle.
- ★ For m minor cycles to k state pipeline, ILP=mk
- ★ CPU Time for n instructions T(n) = (k + (n-1)/m) Tc



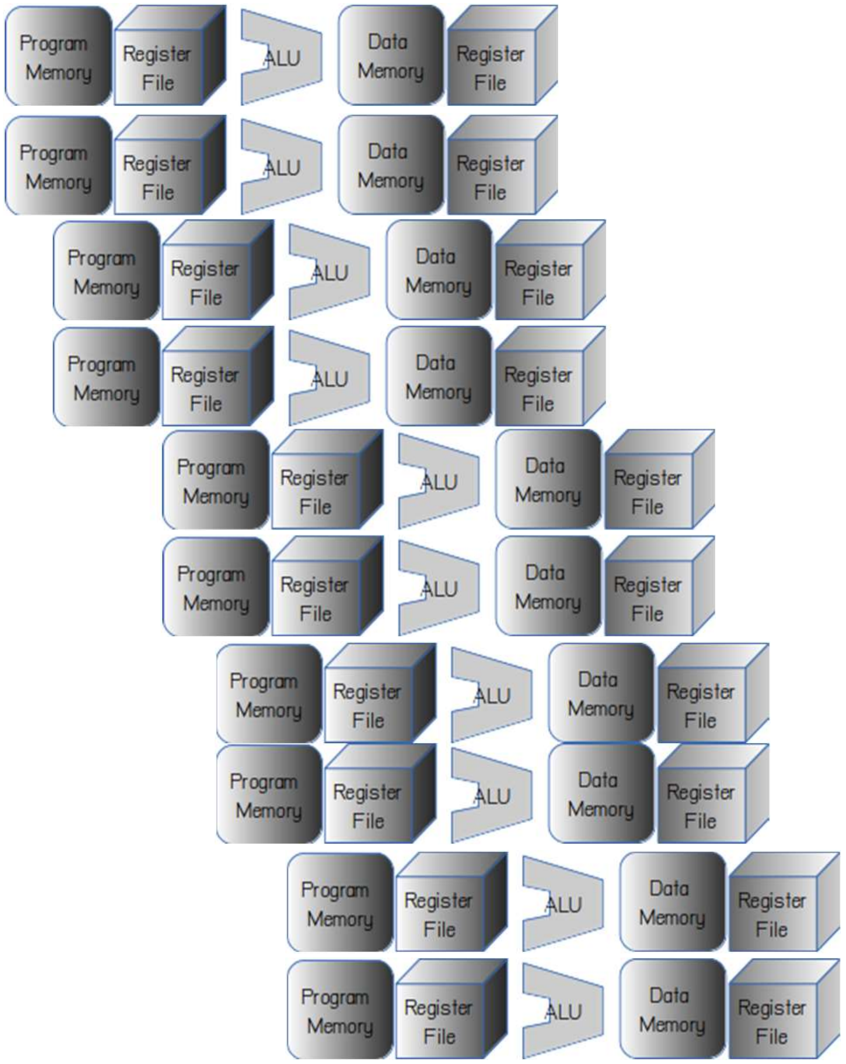
SuperPipeline SuperScalar

★ SuperScalar with minor cycles.

★ In this example, 2 minor cycles is equivalent to 1 major cycle.

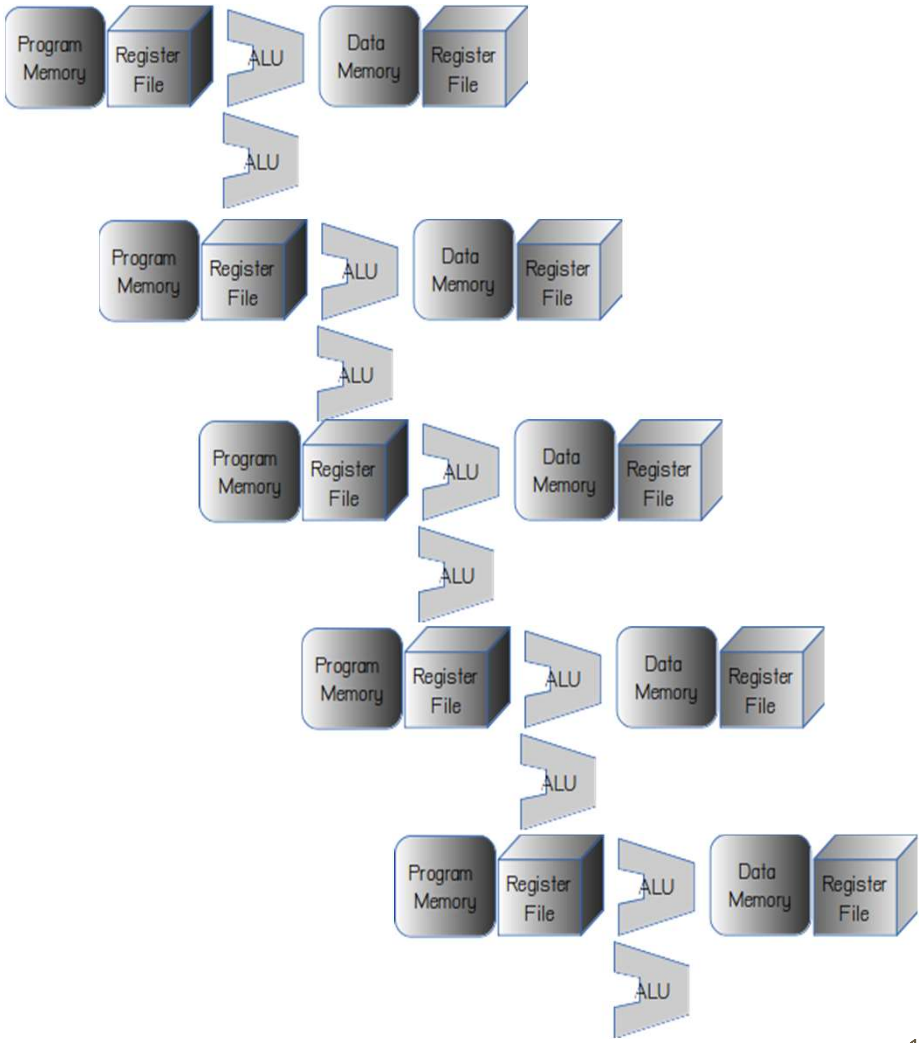
★ For m minor cycles to k states pipeline in superscale degree x,
ILP=mkx

★ CPU Time for n instructions
T(n) = (k + (n-x)/mx) Tc



VLIW

- ★ VLIW with degree 2 and 5 states of pipeline has 10 instructions in flight.
- ★ For k state pipelines in VLIW x degrees,
  ILP = kx.
- ★ CPU Time for n major instructions
  T(n) = (k + (n/x-x)) Tc



VLIW in real world

- ★ Static scheduling.
- ★ Two lines for ordering
- ★ One for rice
- ★ Another for noodle

SuperSymmetry

★ SuperScalar
★ Speedup=
(k + (n-x)/x) Tc /
(k + (n-1)) Tc

★ For large n,
Speedup=x

★ SuperPipeline
★ Speedup=
(k + (n-1)/m) Tc/
(k + (n-1)) Tc

★ For large n,
Speedup=m

★ VLIW
★ Speedup=
(k + (n/x-x)) Tc/
(k + (n-1)) Tc

★ For large n,
Speedup=x

At the same degree, <u>SuperScalar, SuperPipeline and VLIW</u> yield the <u>same</u> performance.

Observations on VLIW

★ Software for Intel Itanium and Intel Itanium II (also the successors) are not binary compatible. **Why?** (Hint. static scheduling)

★ As of January 2019, Intel officially announced the end of life and product discontinuance of the Itanium CPU family.

★ Other implementations of VLIW exist (e.g. Transmeta Crusoe). They are not practically used today.

Speed Up (pipeline)

- ★ Assuming that there are two separate pipelines (integer unit and floating point unit), two instructions can be executed at the sametime.
- ★ CPU Time = Max (Int, FP)
- ★ Potential Speed Up

  =19/Max(Int,FP)

  =19/Max(7,12)

  =1.6

Basic Loop:	Cycles
load Ra <- Ai	1
load Ry <- Yi	1
fmult Rm <- Ra*Rx	7
fadd Rs <- Rm+Ry	5
store Ai <- Rs	1
inc Yi	1
dec i	1
inc Ai	1
branch	1

Performance (Revisit)

★ CPU Time = IC x CPI x TC
★ In practical, operation classes and latency are not balance.

	IC	CPI	TC	Potential Speed Up
Single Cycle Processor	n	1	L	1
Multiple Cycle Processor	n	3-5 (avg=4.2)	L/5	1.xx
Pipelining Processor	n	1	L/5	5
SuperScalar (degree k)	n	1/k	L/5	5k
SuperPipeline (m minor cycle)	n	1/m	L/5	5m
VLIW (x instructions in a word)	n/x	1	L/5	5x

Compiler & ILP

★ Compiler can play a critical role in exploding more ILP
★ Small basic block means less opportunity for filling available slots.
★ A larger basic block means more opportunities for filling slots.
★ To increase a basic block,
  ○ Loop Unrolling
  ○ Software Pipeline

Loop Unroll

★ Make the basic block in a loop larger by having more iterations in a loop.
★ The new code is potentially not stall at the cost of larger code size.

Normal loop.

```
for (i=MAX-1;i>=0;i--) {
    A[i] = A[i] + C;
}
```

Symbolically unroll 4 iterations

```
for (i=MAX-1;i>=3;i-=4) {
    A[i] = A[i] + C;
    A[i-1] = A[i-1] + C;
    A[i-2] = A[i-2] + C;
    A[i-3] = A[i-3] + C;
}
```

Loop Unroll (ctd.)

Normal Loop

```
LOOP:
            LW    $f0, $r1
(stall)
    ADD     $f2, $f0,$f1
(stall)
(stall)
    SW     $f2, $r1
    SUB     $r1, $r1,#4
    BNE     $r1, LOOP
```

8 cycles / Iteration

Register Renaming

- ★ To free unrolled iterations from the first iteration, a register would be renamed.
- ★ Each iteration is now resource independent.

Unroll (4 iterations)

```
LOOP:
   LW     $f0, 0($r1)
(stall)
   ADD     $f2, $f0, $f1
(stall)
(stall)
   SW     $f2, 0($r1)
   LW     $f3, 4($r1)
(stall)
   ADD     $f4, $f3, $f1
(stall)
(stall)
   SW     $f4, 4($r1)
   LW     $f5, 8($r1)
(stall)
   ADD     $f6, $f5, $f1
(stall)
(stall)
   SW     $f6, 8($r1)
   LW     $f7, 12($r1)
(stall)
   ADD     $f8, $f6, $f1
(stall)
(stall)
   SW     $f8, 12($r1)
   SUB     $r1, $r1,#16
   BNE     $r1, LOOP
```

Unroll + Rescheduling

```
LOOP:
    LW     $f0, 0($r1)
    LW     $f3, 4($r1)
    LW     $f5, 8($r1)
    LW     $f7, 12($r1)
    ADD     $f2, $f0, $f1
    ADD     $f4, $f3, $f1
    ADD     $f6, $f5, $f1
    ADD     $f8, $f6, $f1
    SW     $f2, 0($r1)
    SW     $f4, 4($r1)
    SW     $f6, 8($r1)
    SW     $f8, 12($r1)
    SUB     $r1, $r1,#16
    BNE     $r1, LOOP
```

14 cycles / 4 Iteration
= 3.5 cycles per iteration

Speed Up=8/3.5

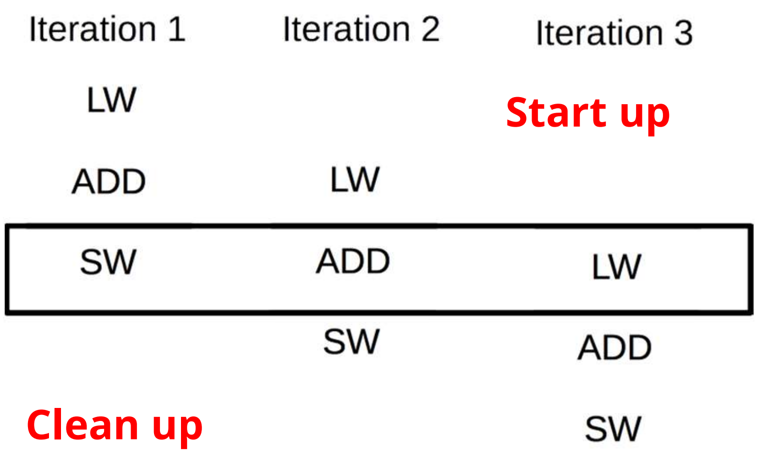
Loop Unroll and Superscalar

12 cycles / 5 Iteration
= 2.4 cycles per iteration

Integer Unit	Floating Unit	Clock Cycle (Issued)
LOOP:		
LW **$f0**, 0($r1)		1
LW $f3, 4($r1)		2
LW $f5, 8($r1)	ADD $f2, **$f0**, $f1	3
LW $f7, 12($r1)	ADD $f4, $f3, $f1	4
LW $f9, 16($r1)	ADD $f6, $f5, $f1	5
SW $f2, 0($r1)	ADD $f8, $f6, $f1	6
SW $f4, 4($r1)	ADD $f10, $9, $f1	7
SW $f6, 8($r1)		8
SW $f8, 12($r1)		9
SW $f10, 16($r1)		10
SUB $r1, $r1,#24		11
BNE $r1, LOOP		12

Software Pipeline

★ Unroll as needed only.
★ Good performance, appropriate code size.



Unroll (3 terations)

```
LOOP:
    LW     $f0, 0($r1)
    LW     $f3, 4($r1)
    LW     $f5, 8($r1)
    ADD     $f2, $f0, $f1
    ADD     $f4, $f3, $f1
    ADD     $f6, $f5, $f1
    SW     $f2, 0($r1)
    SW     $f4, 4($r1)
    SW     $f6, 8($r1)
    SUB     $r1, $r1, #12
    BNE     $r1, LOOP
```

Unroll + Rescheduling

Start up

```
    SUB     $r1, $r1, #4
    LW     $f0, 4($r1)
    ADD     $f2, $f0, $f1
    LW     $f0, 0($r1)
LOOP:
    SW     $f2, 0($r1)
    ADD     $f2, $f0, $f1
    LW     $f0, 8($r1)
    SUB     $r1, $r1, #4
    BNE     $r1, LOOP

    SW     $f2, 4($r1)
    ADD     $f2, $f0, $f1
    SW     $f2, 0($r1)
```

Clean up

Dynamic vs. Static Scheduling

- ★ **Dynamic Scheduling (Hardware)**
- ★ Hardware dynamically schedules code (out-of-order execution).
  - ○ Loop are dynamically unrolled in the processor.
  - ○ Multiple instructions are dynamically issued.
- ★ Legacy software gains better performance on new hardware.
- ★ Complex hardware design?

- ★ **Static Scheduling (Software/Compiler)**
- ★ Compiler can see larger scope.
  - ○ Loop are statically unrolled by compilers.
  - ○ Multiple instructions are statically scheduled for each cycle (see VLIW).
- ★ Poor compatibilities
- ★ Dependency cannot be determined at compile time.
- ★ Simple hardware
- ★ Lower Power requirements?

Who is the winner?

How about a chip with multicores?

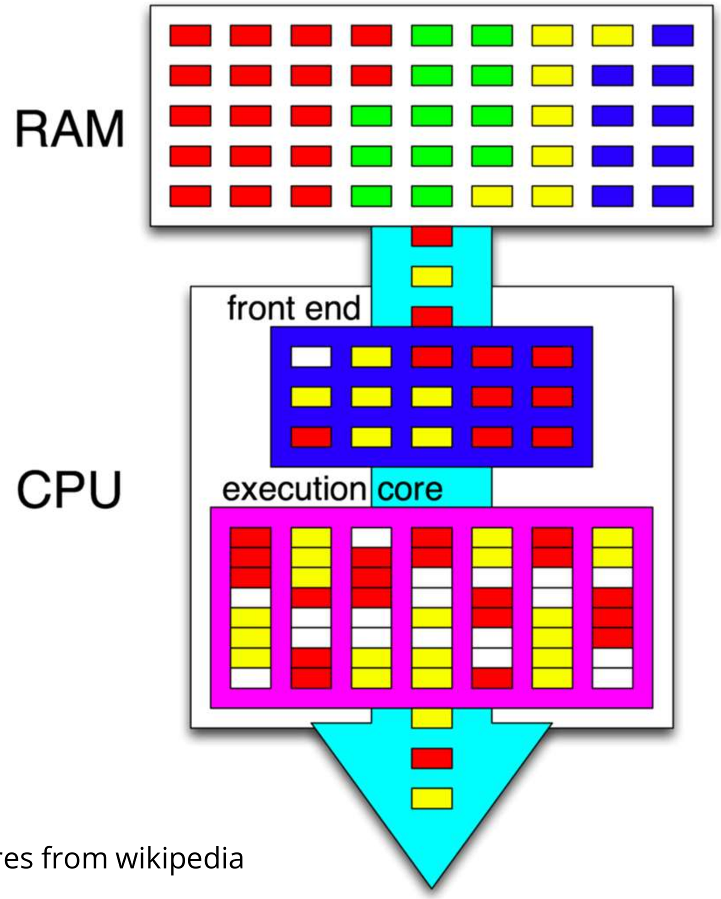
Chip with multicores

- ★ A chip with multicores is basically a chip with multiple superscalar cores inside.
- ★ Can also be viewed as Multiple Pipes.
- ★ Demands for more ILP. -- Thread-Level parallelism

Thread-Level Parallelism

The detail is beyond the scope of this course.
To learn more, join
High Performance Architecture.

★ Since a program (thread) cannot fill available slots in the processor, let's get instructions from several threads.
★ Instructions in pipeline are coming from several threads.
★ The technology is called simultaneous multithreading (SMT).
★ Intel implementation is officially called Hyper-Threading Technology.



Pictures from wikipedia

Summary

- ★ There exist several approaches to ILP.
- ★ At similar degrees, SuperScalar, SuperPipeline, VLIW will yield similar performance.
- ★ Dynamic scheduling allows legacy software to gain performance.
- ★ Static scheduling may gain more benefits if a compiler is more powerful.
    - ○ Proven to be too difficult to have good compiler.

Exercises

Loop Unroll

- ★ For the given code, if MAX is a known/large value,
  (1)how many iterations can we unroll this loop?
  (2) What is the potential speed up of the unrolled (rescheduled) code comparing to the original one.
  (3) What will be the size of the unrolled code?
  (4) Which resources is limited to the number of iterations that can be unrolled?

```
for (i=MAX-1;i>=0;i--) {
    A[i] = A[i] + C;
}
```

```
LOOP:
             LW    $f0,
$r1
(stall)
    ADD    $f2, $f0,$f1
(stall)
(stall)
    SW    $f2, $r1
    SUB    $r1, $r1,#4
    BNE    $r1, LOOP
```

Software Pipeline

★ For the given code, if MAX is a known/large value, please use software pipeline to optimize the code.
(1) What is the potential speed up of the unrolled (rescheduled) code comparing to the loop unroll version.
(2) What will be the size of the code?

```
for (i=MAX-1;i>=0;i--) {
    A[i] = A[i] + C;
}
```

```
LOOP:
              LW    $f0,
$r1
(stall)
    ADD    $f2, $f0,$f1
(stall)
(stall)
    SW    $f2, $r1
    SUB    $r1, $r1,#4
    BNE    $r1, LOOP
```

Dynamic SuperScalar (out-of-order execution)

- ★ Given a super scalar at degree two, please draw a diagram showing the pipeline of the following code.

```
LOOP:
    LW     $f0, $r1
    ADD     $f2, $f0,$f1
    SW     $f2, $r1
    SUB     $r1, $r1,#4
    BNE     $r1, LOOP
```

		Issue	EX	MEM	WB	Notes
LW	$f0, $r1	1	2	3	4	
ADD	$f2, $f0,$f1	1	5,6,7		8	Wait $f0
SW	$f2, $r1	2	3	9		Wait $f2
SUB	$r1, $r1,#4	2	4		5	
BNE	$r1, LOOP	3	6			
LW	$f0, $r1	4	7	8	9	
ADD	$f2, $f0,$f1	4	10,11,12		13	Wait $f0
SW	$f2, $r1	5	8	14		
SUB	$r1, $r1,#4	5	9		10	
BNE	$r1, LOOP	6	11			

End of Chapter 7