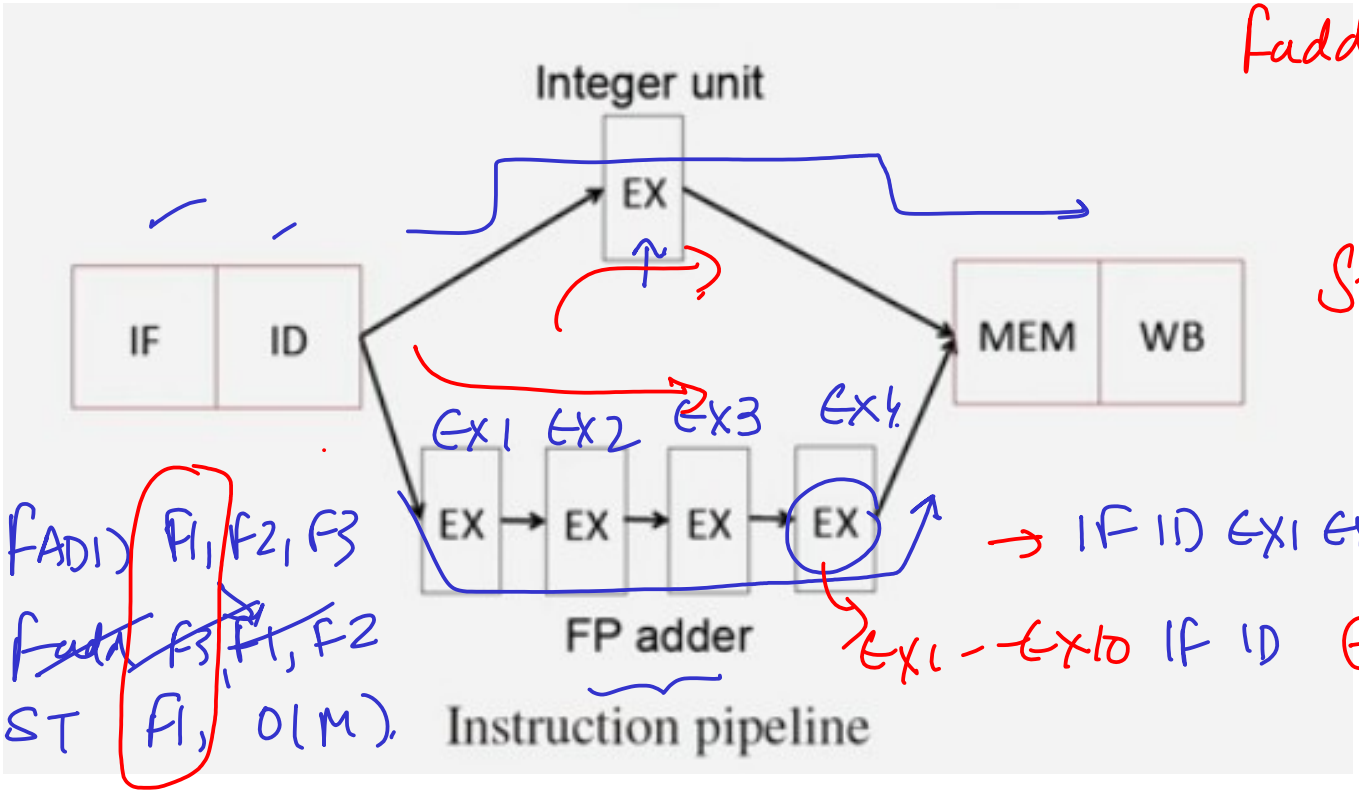
Compiler technqiues to improve ILP

Instruction level parallelism

- Static scheduling: Re-ordering
- Loop unrolling



fadd IF ID - EX1 - EX4 - M - W

STORE IF ID EX M W

EX1 EX2 EX3 EX4

→ FADD F1, F2, F3

~~fadd F3, F1, F2~~

→ ST F1, 0(M).

→ IF ID EX1 EX2 EX3 EX4 M W

EX1 - EX10 IF ID EX - - M. W

Value Producing instruction	Consuming instruction	Pipeline Latency
FP ALU op	Another FP ALU op	3
FP ALU op (path2)	Store double (path 1)	~~2~~ 3
Load double	FP ALU op	1
Load double	Store double	0

LD

FP

2 stalls

1 - str haz.

Loop unrolling

```
for (i=999; i>=0; i --)
   x[i] = x[i] + y;
```

Operation on floating point doubles--> 8 bytes of space in memory

Loop unrolling

```
for (i=999; i>=0; i --)
   x[i] = x[i] + y;
```

array memory

fadd.

Compile

```
Loop:
L.D   F0, 0(R1)  // i = R1 = integer reg. F0 = floating pt reg
FADD.D   F4, F0, F2    //y =  F2
S.D   F4, 0(R1)
SUBI    R1, R1, #8
BNEQ  R1,R2 Loop  //R2=0
```

Dependencies?

Loop unrolling

Dependencies?
And number of stalls?

```
for (i=999; i>=0; i --)
   x[i] = x[i] + y;
```

```
Loop:
L.D   F0, 0(R1)  // i = R1 = integer reg. F0 = floating pt reg
ADD.D     F4, F0, F2    //y =  F2 – floating pt adder pipeline
S.D  F4, 0(R1)
SUBI    R1, R1, #8
BNEQ  R1,R2 Loop  //R2=0
```

Stalls

LD	IF	ID	EX	M	W						
Fadd		IF	ID	--	EX1	EX2	EX3	EX4	M	W	
store				IF	ID	--	--	EX	--	M	W

Structural hazard

Loop:

L.D F0, 0(R1)

Stall

ADD.D F4, F0, F2 //y = F2

2 extra stalls +1 stall

due to structural hazard

Reorder?

S.D F4, 0(R1)

SUBI R1, R1, #8

Stall --> fast branching for Branch / decide in ID

BNEQ R1,R2 Loop //R2=0

Sub IF ID EX M W

R1

Bneq IF ID EX M W

IF - ID EX M W

Re-ordering/Scheduling

```
Loop:
L.D   F0, 0(R1)
Stall
ADD.D     F4, F0, F2    //y =  F2
3 stalls
S.D  F4, 0(R1)
SUBI    R1, R1, #8
Stall
BNEQ  R1,R2 Loop
```

1 independent instr

```
Loop:
L.D   F0, 0(R1)
SUBI    R1, R1, #8
ADD.D     F4, F0, F2
3 stalls
S.D  F4, 0(R1)
BNEQ  R1,R2 Loop
```

5 * 1000 = 5000 instructions, 2 stalls

Re-ordering/Scheduling

```
Loop:
L.D   F0, 0(R1)
Stall
ADD.D     F4, F0, F2    //y =  F2
3 stalls
S.D  F4, 0(R1)
SUBI    R1, R1, #8
Stall
BNEQ  R1,R2 Loop
```

→

```
Loop:
L.D   F0, 0(R1)
SUBI    R1, R1, #8
ADD.D     F4, F0, F2
3 stalls
S.D  F4, 8(R1)
BNEQ  R1,R2 Loop
```

5 * 1000 = 5000 instructions, 2 stalls

Loop unrolling

```
for (i=999; i>=0; i--)
  x[i] = x[i] + y;
```

Unroll loop once

```
for (i=999; i>=0; i=i-2)
{
  x[i] = x[i] + y;
  x[i-1] = x[i-1] + y;
}
```

Half the iterations

Loop unrolling

```
for (i=999; i>=0; i=i-2)
{
  x[i] = x[i] + y;
  x[i-1] = x[i-1] + y;
}
```

Reg renaming

Reordering

??

```
Loop:
L.D   F0, 0(R1)
ADD.D     F4, F0, F2
S.D  F4, 0(R1)
L.D  F0, 0(R1)
ADD.D   F4, F0, F2
S.D  F4, 0(R1)
SUBI    R1, R1, #8
BNEQ  R1,R2 Loop
```

stall

3

1

3

Unroll once

```
for (i=999; i>=0; i=i-2)
{
   x[i] = x[i] + y;
   x[i-1] = x[i-1] + y;
}
```

Ld x(i)

x(i+)

```
Loop:
L.D   F0, 0(R1)
ADD.D     F4, F0, F2
S.D  F4, 0(R1)
L.D  F6, -8(R1)
ADD.D     F8, F6, F2
S.D  F8, -8(R1)
SUBI    R1, R1, 16
BNEQ  R1,R2 Loop
```

1st →

3st →

Increase in instructions ?

Unroll once --> half the iterations

```
for (i=999; i>=0; i=i-2)
{
   x[i] = x[i] + y;
   x[i-1] = x[i-1] + y;
}
```

$x(i-2) = x(i-2) + y$

Ld x(i)
x(i-1) } load

x(i) + y
x(i-1) + y } add

```
Loop:
L.D   F0, 0(R1)
ADD.D     F4, F0, F2
S.D  F4, 0(R1)
L.D   F6, -8(R1)
ADD.D     F8, F6, F2
S.D  F8, -8(R1)
SUBI    R1, R1, 16
BNEQ  R1,R2 Loop
```

Ld
add
Sd

8 * 500 = 4000 instructions
Stalls?

stalls

```
L.D     F0, 0(R1)
Stall
ADD.D   F4, F0, F2
3 stalls
S.D     F4, 0(R1)
L.D     F6, -8(R1)
Stall
ADD.D   F8, F6, F2
3 stalls
S.D     F8, -8(R1)
SUBI    R1, R1, 16
Stall
BNEQ    R1,R2 Loop
```

$x(i)$

$x(i-1)$

Re-ordering/Scheduling

```
L.D     F0, 0(R1)
Stall
ADD.D   F4, F0, F2
3 stalls
S.D     F4, 0(R1)
L.D     F6, -8(R1)
Stall
ADD.D   F8, F6, F2
3 stalls
S.D     F8, -8(R1)
SUBI    R1, R1, 16
Stall
BNEQ R1,R2 Loop
```

2 +1 str.

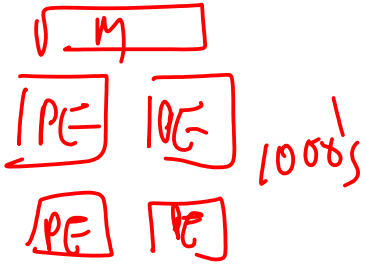
1 instr

```
L.D     F0, 0(R1)
L.D     F6, -8(R1)      -->
renaming useful here
ADD.D   F4, F0, F2
ADD.D   F8, F6, F2
SUBI    R1, R1, 16
stall
S.D     F4, 16(R1)
S.D     F8, 8(R1)
BNEQ    R1,R2 Loop
```

2 stalls → 1

Reduced stalls--> better CPI
Reduced instructions 8 * 500 = 4000
Exec time = Instr * CPI * clk cycle time

Unroll thrice




```
for (i=999; i>=0; i=i-2)
{
   x[i] = x[i] + y;
   x[i-1] = x[i-1] + y;
   x[i-2] = x[i-2] + y;
   x[i-3] = x[i-3] + y;
}
```

Need to change reg names --> 16 stalls

Twice

```
L.D     F0, 0(R1)
ADD.D   F4, F0, F2
S.D     F4, 0(R1)
L.D     F6, -8(R1)
ADD.D   F8, F6, F2
S.D     F8, -8(R1)
L.D     F0, 0(R1)
ADD.D   F4, F0, F2
S.D     F4, 0(R1)
L.D     F0, 0(R1)
ADD.D   F4, F0, F2
S.D     F4, 0(R1)
SUBI    R1, R1, 16
BNEQ    R1,R2 Loop
```

Schedule/Re-order
No stalls

```
L.D     F0, 0(R1)
L.D     F6, -8(R1)
Load
Load
ADD.D   F4, F0, F2
ADD.D   F8, F6, F2
Add
Add
SUBI    R1, R1, 16
S.D     F4, 16(R1)
S.D     F8, 8(R1)
Store
Store
BNEQ    R1,R2 Loop
```

100

GPU
TPU
FPGA

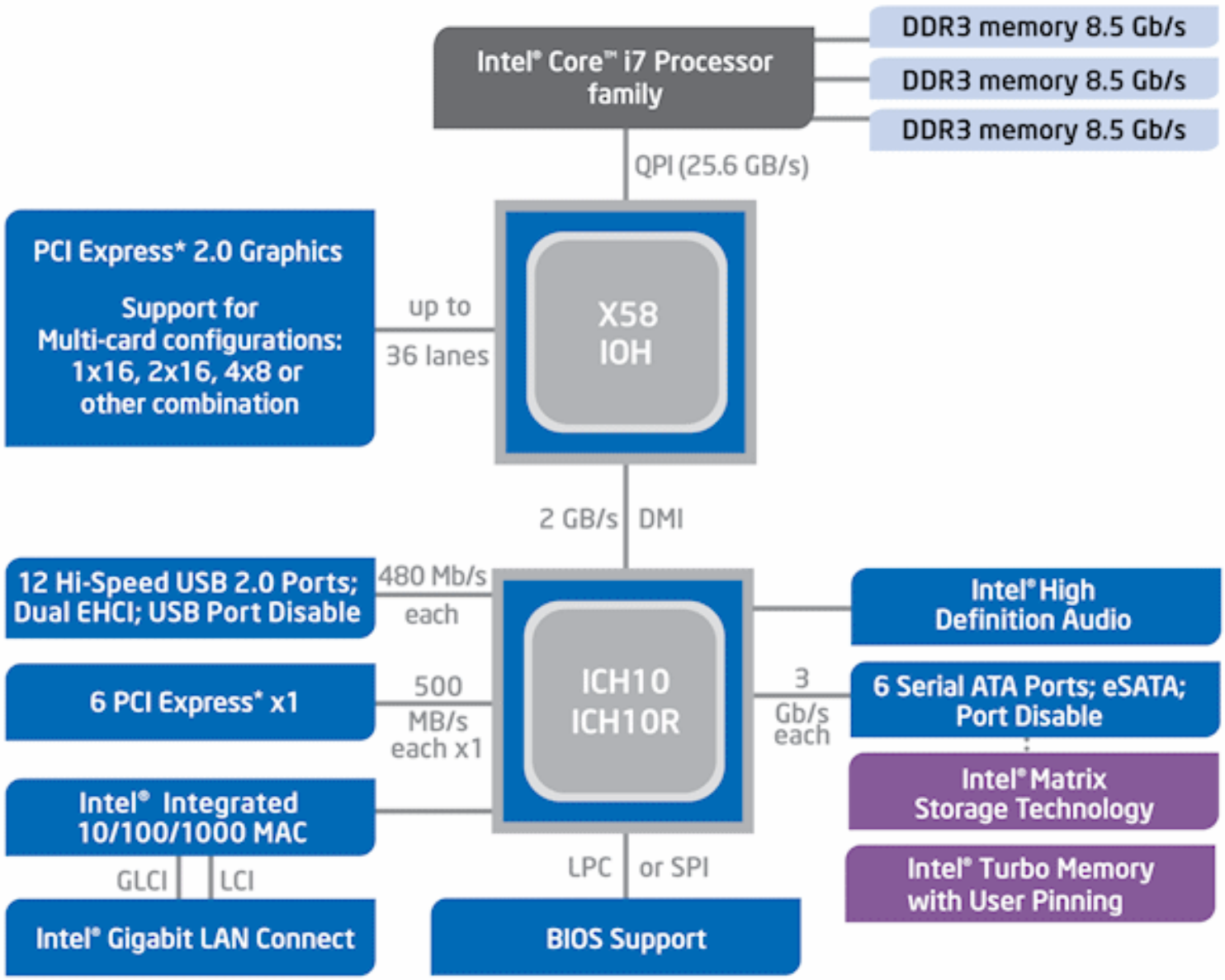
MAC

a*b+c

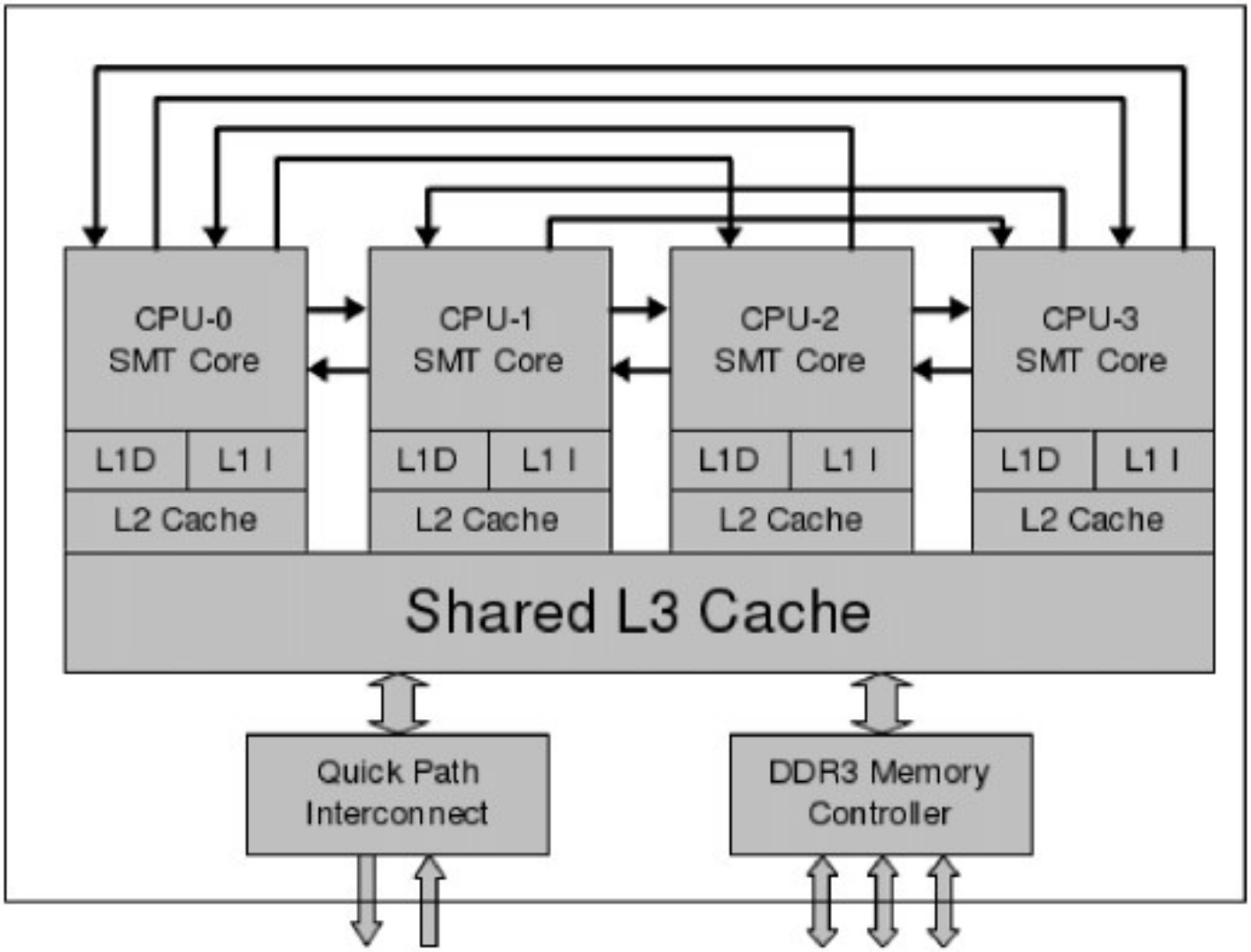
P - General Purpose

Cache

Memory.

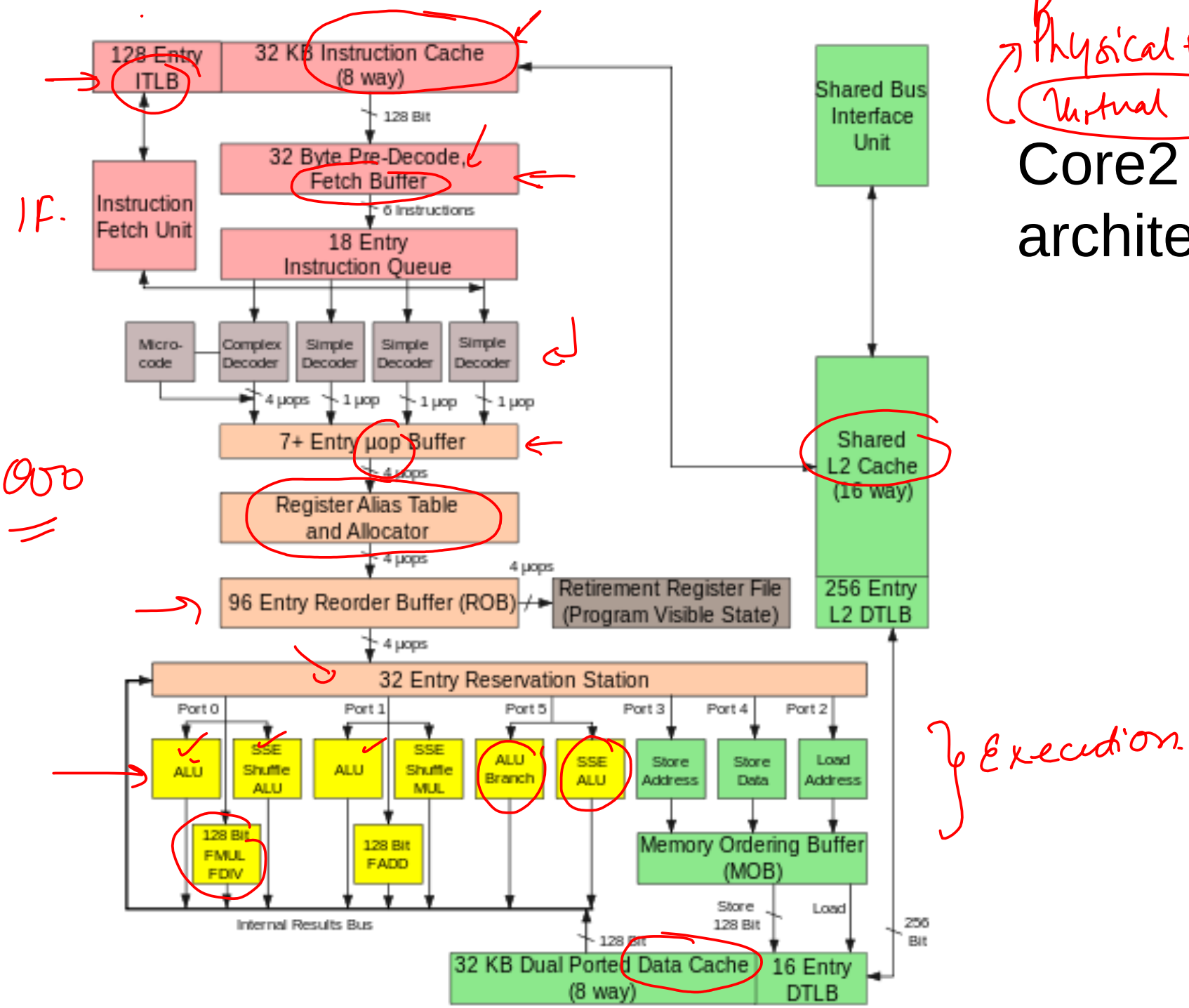


http://onlyusama.blogspot.com/2009/06/block-diagram-of-core-i7-system.html

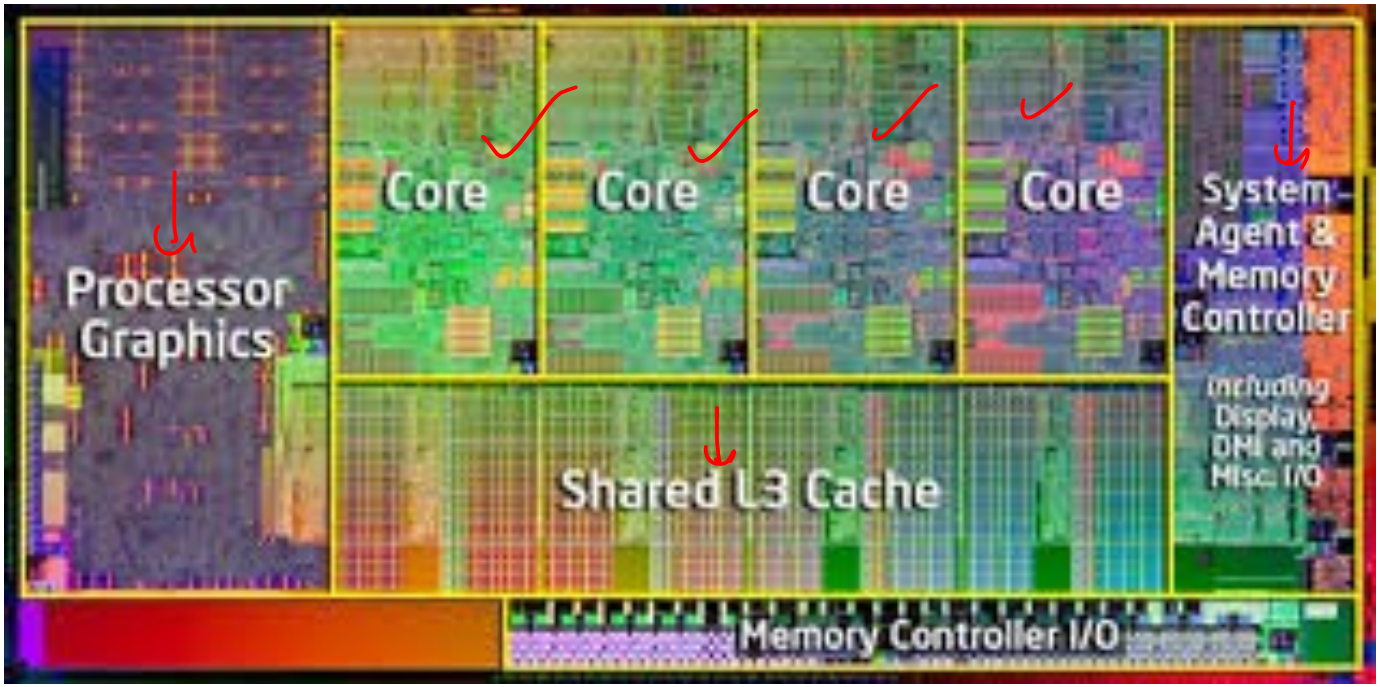


Evaluating Multicore Architectures for Application in High Assurance Systems - ResearchGate

Core2 architecture



Intel Core 2 Architecture



Dec 6 Caches Dec 7 Dec 13

Cache

Protocol

Pages

RAM

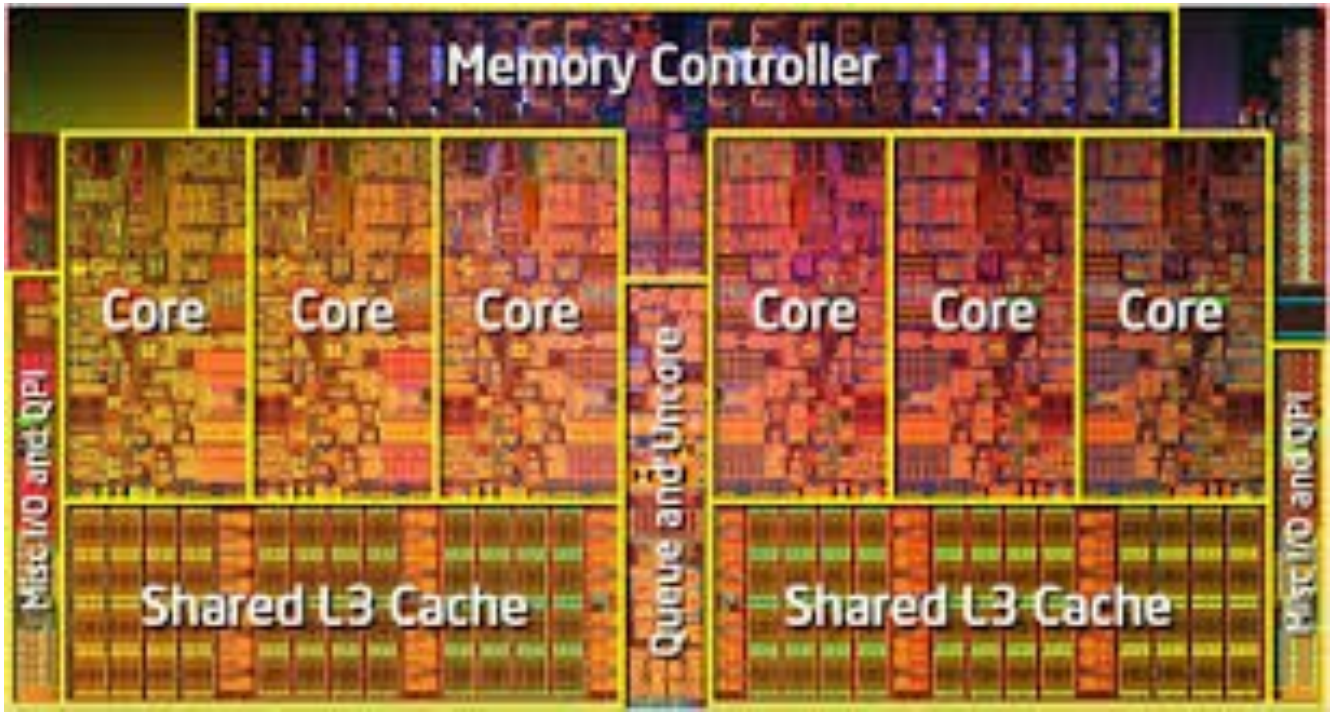
16GB 1TB

Cache

Exceptions

Multi-core

cachecoherency



https://www.cs.uaf.edu/2009/fall/cs441/proj1/russell/index.html

Acknowledgements

- Compiler Optimizations for Exposing ILP – IIT Madras
- Loop Unrolling Benefits - Georgia Tech