C66x CorePac: Achieving High Performance

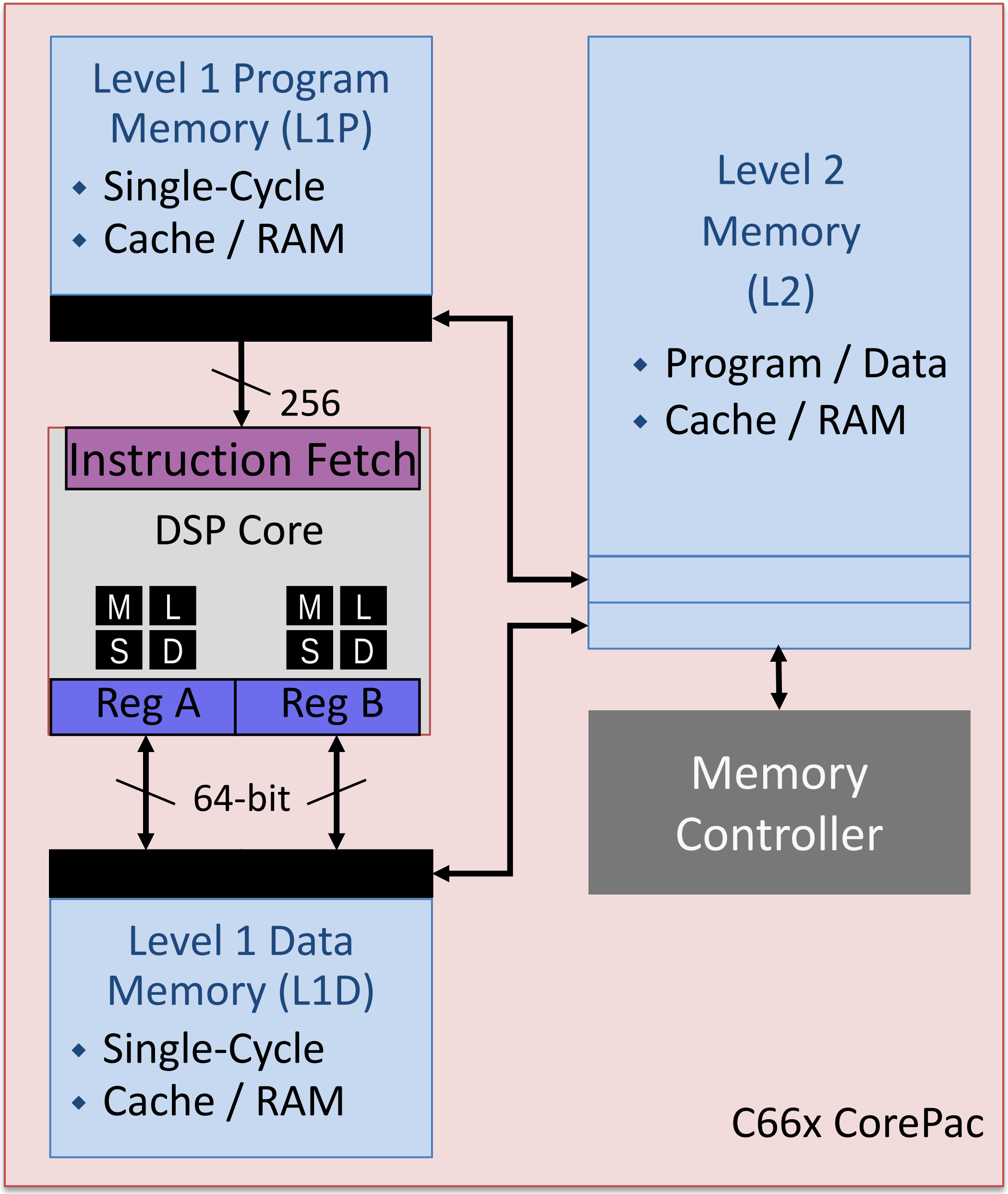
Agenda

1. CorePac Architecture
2. Single Instruction Multiple Data (SIMD)
3. Memory Access
4. Pipeline Concept

CorePac Architecture

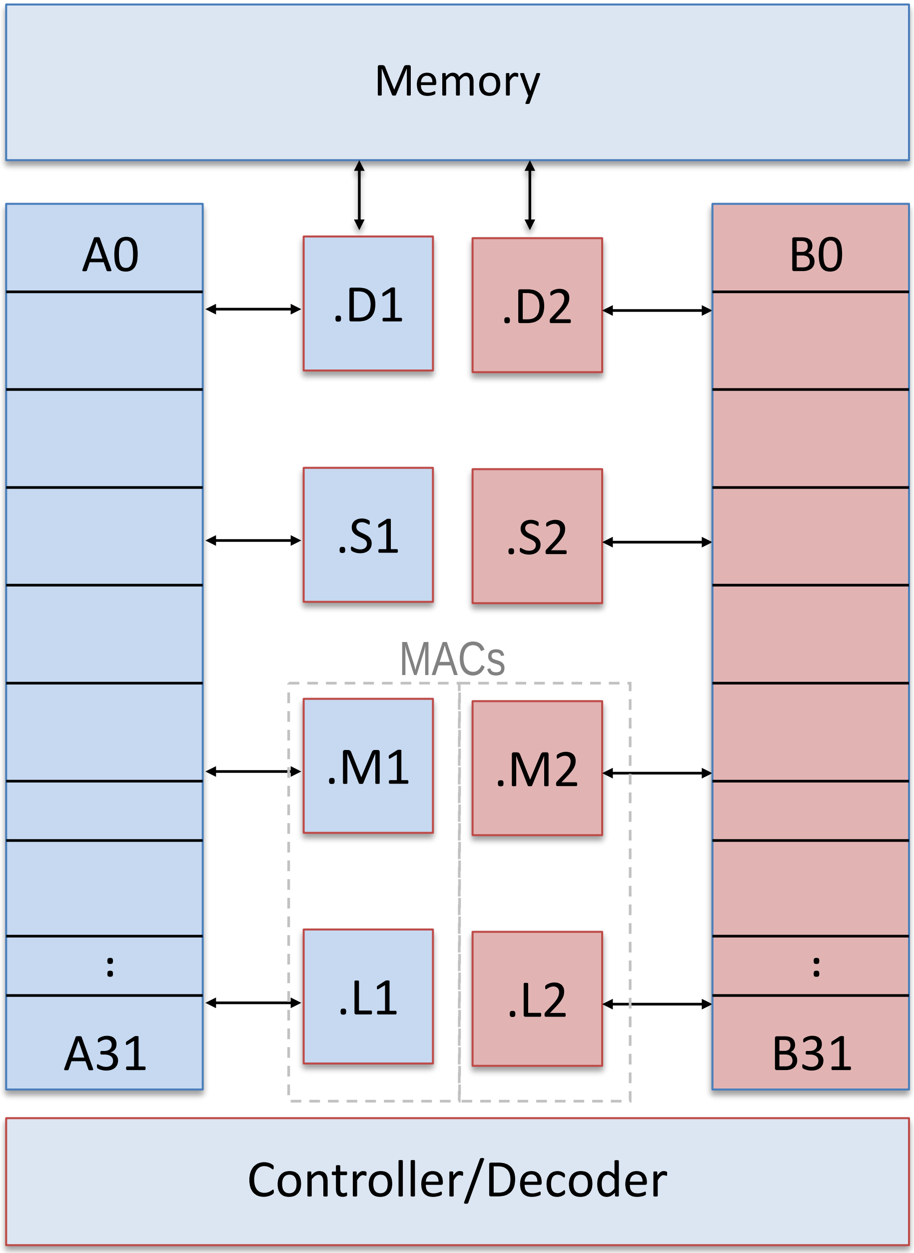
1. CorePac Architecture
2. Single Instruction Multiple Data (SIMD)
3. Memory Access
4. Pipeline Concept

C66x CorePac



CorePac includes:

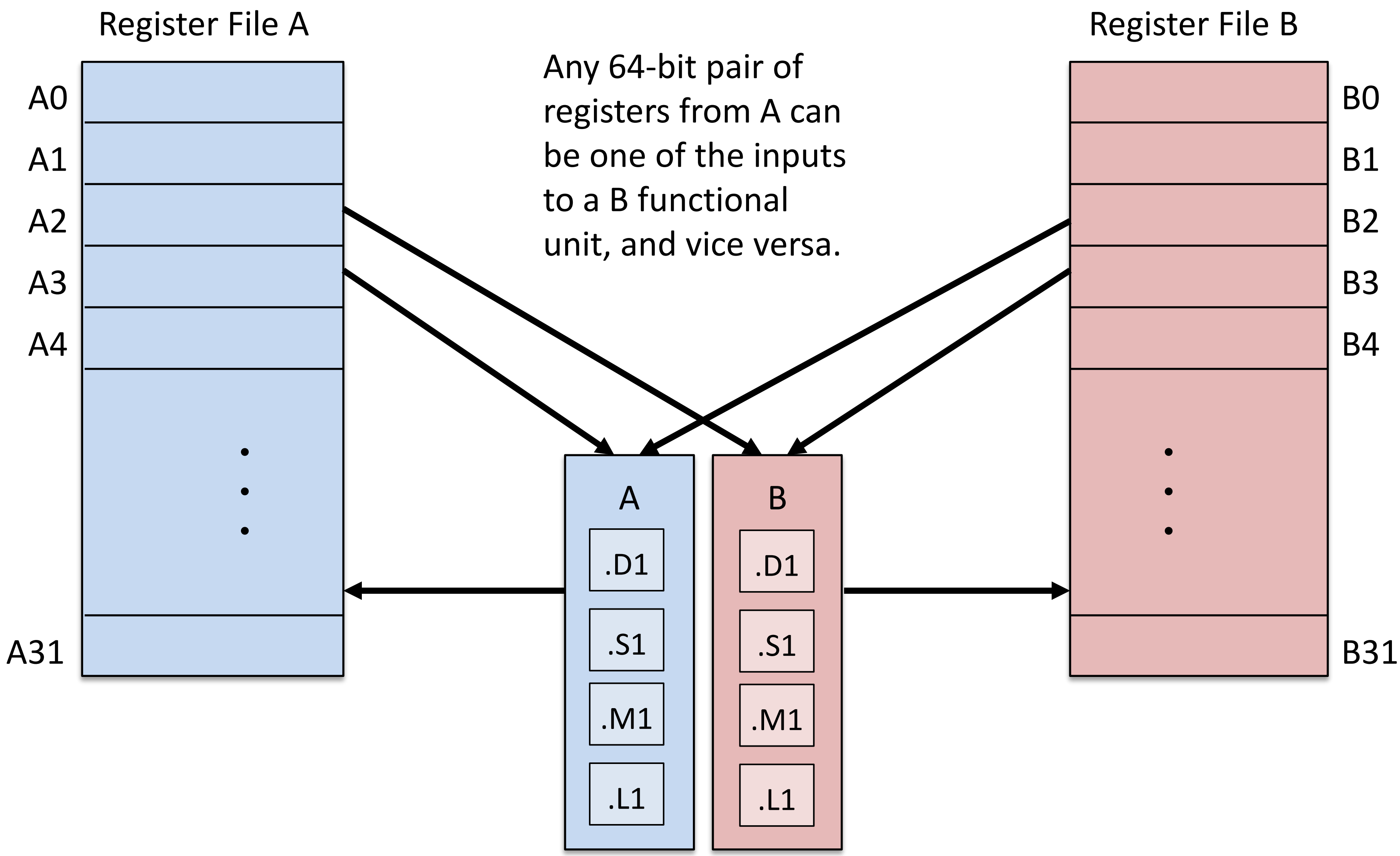
- DSP Core
  - Two register sets
  - Four functional units per register side
- L1P memory (Cache/RAM)
- L1D memory (Cache/RAM)
- L2 memory (Cache/RAM)



C66x DSP Core

- Four functional units per side:
  - Multiplier (.M)
  - ALU (.L)
  - Data (.D)
  - Control (.S)
- These independent functional units enable efficient execution of parallel specialized instructions:
  - Multiplier (.M1and.M2) and ALU (.L1 and .L2) provide MAC (multiple accumulation) operations.
  - Data (.D) provides data input/output.
  - Control (.S) provides control functions (loop, branch, call).
- Each DSP core dispatches up to eight parallel instructions each cycle.
- All instructions are conditional, which enables efficient pipelining.
- The optimized C compiler generates efficient target code.

C66x DSP Core Cross-Path



Partial List of .D Instructions

Table C-1 Instructions Executing in the .D Functional Unit

Instruction	Instruction
ADD	OR
ADDAB	STB
ADDAD	STB[1] (15-bit offset)
ADDAH	STDW
ADDAW	STH
ADD2	STH[1] (15-bit offset)
AND	STNDW
ANDN	STNW
LDB and LDB(U)	STW
LDB and LDB(U)[1] (15-bit offset)	STW[1] (15-bit offset)
LDDW	SUB
LDH and LDH(U)	SUBAB
LDH and LDH(U)[1]	SUBAH
LDNDW	SUBAW
LDNW	SUB2
LDW	XOR
LDW[1] (15-bit offset)	ZERO
MV	
MVK	

1. D2 only

Partial List of .L Instructions

Table D-1 Instructions Executing in the .L Functional Unit

Instruction	Instruction	Instruction	Instruction
ABS	DPACK2	NORM	SPTRUNC
ABS2	DPACKX2	NOT	SSUB
ADD	DPINT	OR	SSUB2
ADDDP	DPSP	PACK2	SUB
ADDSP	DPTRUNC	PACKH2	SUBABS4
ADDSUB	INTDP	PACKH4	SUBC
ADDSUB2	INTDPU	PACKHL2	SUBDP
ADDU	INTSP	PACKLH2	SUBSP
ADD2	INTSPU	PACKL4	SUBU
ADD4	LMBD	SADD	SUB2
AND	MAX2	SADDSUB	SUB4
ANDN	MAXU4	SADDSUB2	SWAP2
CMPEQ	MIN2	SAT	SWAP4
CMPGT	MINU4	SHFL3	UNPKHU4
CMPGTU	MV	SHLMB	UNPKLU4
CMPLT	MVK	SHRMB	XOR
CMPLTU	NEG	SPINT	ZERO

Partial List of .M Instructions

Table E-1 Instructions Executing in the .M Functional Unit

Instruction	Instruction	Instruction	Instruction
AVG2	DOTPUS4	MPYIL	MPY32 (32-bit result)
AVGU4	DOTPU4	MPYILR	MPY32 (64-bit result)
BITC4	GMPY	MPYLH	MPY32SU
BITR	GMPY4	MPYLHU	MPY32U
CMPY	MPY	MPYLI	MPY32US
CMPYR	MPYDP	MPYLIR	MVD
CMPYR1	MPYH	MPYLSHU	ROTL
DDOTP4	MPYHI	MPYLUHS	SHFL
DDOTPH2	MPYHIR	MPYSP	SMPY
DDOTPH2R	MPYHL	MPYSPDP	SMPYH
DDOTPL2	MPYHLU	MPYSP2DP	SMPYHL
DDOTPL2R	MPYHSLU	MPYSU	SMPYLH
DEAL	MPYHSU	MPYSU4	SMPY2
DOTP2	MPYHU	MPYU	SMPY32
DOTPN2	MPYHULS	MPYU4	SSHVL
DOTPNRSU2	MPYHUS	MPYUS	SSHVR
DOTPNRUS2	MPYI	MPYUS4	XORMPY
DOTPRSU2	MPYID	MPY2	XPND2
DOTPRUS2	MPYIH	MPY2IR	XPND4
DOTPSU4	MPYIHR		

Partial List of .S Instructions

Table F-1 Instructions Executing in the .S Functional Unit

Instruction	Instruction	Instruction	Instruction
ABSDP	CMPEQ2	MVKH/MVKLH	SET
ABSSP	CMPEQ4	MVKL	SHL
ADD	CMPEQDP	MVKH/MVKLH	SHLMB
ADDDP	CMPEQSP	NEG	SHR
ADDK	CMPGT2	NOT	SHR2
ADDKPC[1]	CMPGTDP	OR	SHRMB
ADDSP	CMPGTSP	PACK2	SHRU
ADD2	CMPGTU4	PACKH2	SHRU2
AND	CMPLT2	PACKHL2	SPACK2
ANDN	CMPLTDP	PACKLH2	SPACKU4
B displacement	CMPLTSP	RCPDP	SPDP
B register[1]	CMPLTU4	RCPSP	SSHL
B IRP[1]	DMPYU4	RPACK2	SUB
B NRP[1]	EXT	RSQRDP	SUBDP
BDEC	EXTU	RSQRSP	SUBSP
BNOP displacement	MAX2	SADD	SUB2
BNOP register	MIN2	SADD2	SWAP2
BPOS	MV	SADDSU2	UNPKHU4
CALLP	MVC[1]	SADDUS2	UNPKLU4
CLR	MVK	SADDU4	XOR
			ZERO

Single Instruction Multiple Data (SIMD)

1. CorePac Architecture
2. Single Instruction Multiple Data (SIMD)
3. Memory Access
4. Pipeline Concept

C66x SIMD Instructions: Examples

- ADDDP – Add Two Double-Precision Floating-Point Values
- DADD2 – 4-Way SIMD Addition, Packed Signed 16-bit
  - Performs four additions of two sets of four 16-bit numbers packed into 64-bit registers.
  - The four results are rounded to four packed 16-bit values
  - unit = .L1, .L2, .S1, .S2

- FMPYDP - Fast Double-Precision Floating Point Multiply
- QMPY32 - 4-Way SIMD Multiply, Packed Signed 32-bit.
  - Performs four multiplications of two sets of four 32-bit numbers packed into 128-bit registers.
  - The four results are packed 32-bit values.
  - unit = .M1 or .M2

C66x SIMD Instruction: CMATMPY

Many applications use complex matrix arithmetic.

- CMATMPY – 2x1 Complex Vector Multiply 2x2 Complex Matrix
  - Results in 2x1 signed complex vector.
  - All values are 16-bit (16-bit real/16-bit Imaginary)
  - unit = .M1 or .M2

- How many multiplications are complex multiplication, where each complex multiplication has the following:
  - 4 complex multiplications (4 real multiplications each)
  - Two M units (16 multiplications each) = 32 multiplications
  - Core cycles per second (1.25 G)
  - Total multiplications per second = 40 G multiplications
  - 8 cores = 320 G multiplications

The issue here is, can we feed the functional units data fast enough?

Feeding the Functional Units

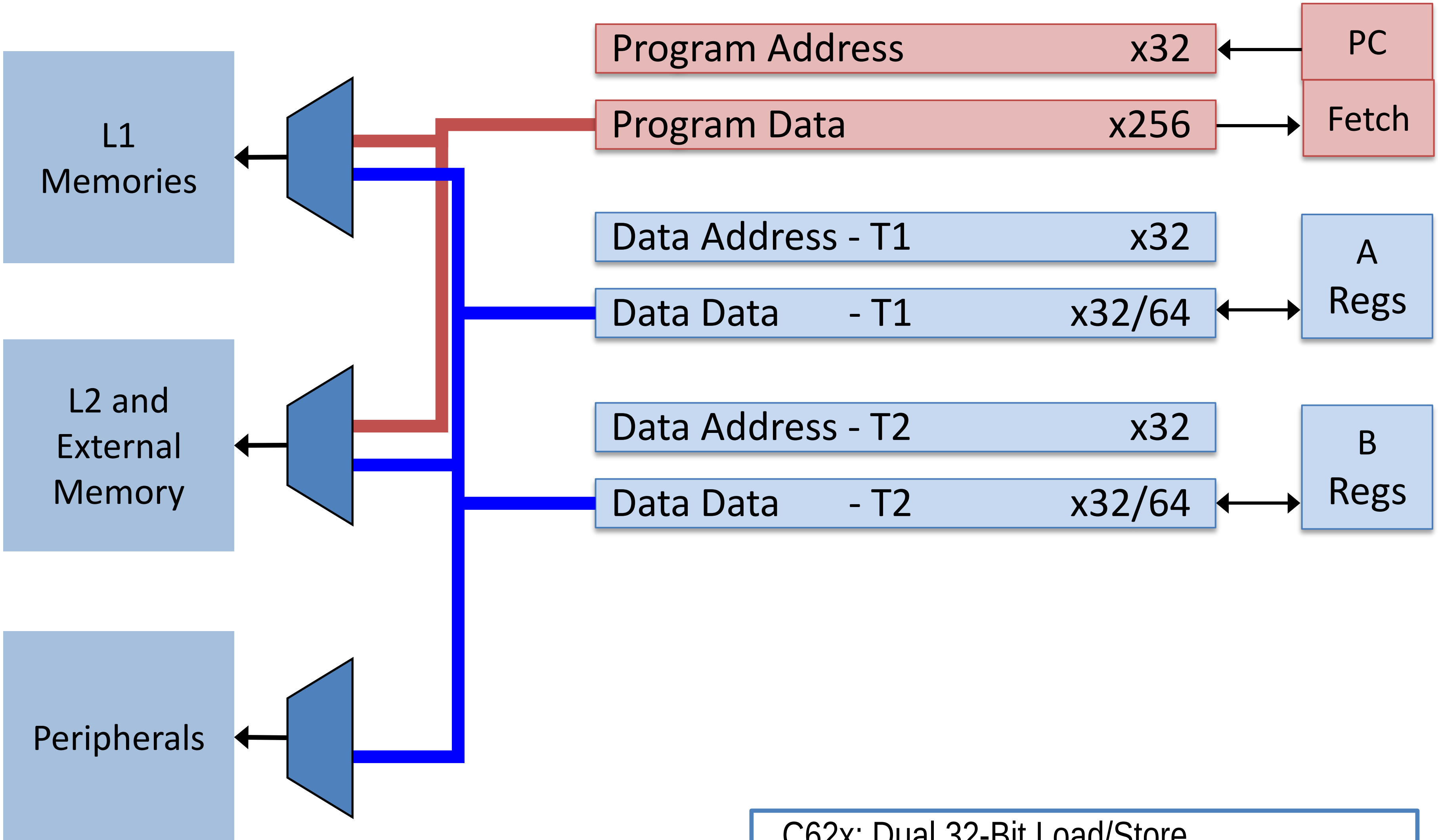
There are two challenges:

- How to provide enough data from memory to the core
  - Access to L1 memory is wide (2 x 64 bit) and fast (0 wait state)
  - Multiple mechanisms are used to efficiently transfer new data to L1 from L2 and external memory.
- How to get values in and out of the functional units
  - Hardware pipeline enables execution of instructions every cycle.
  - Software pipeline enables efficient instruction scheduling to maximize functional unit throughput.

Memory Access

1. CorePac Architecture
2. Single Instruction Multiple Data (SIMD)
3. Memory Access
4. Pipeline Concept

Internal Buses

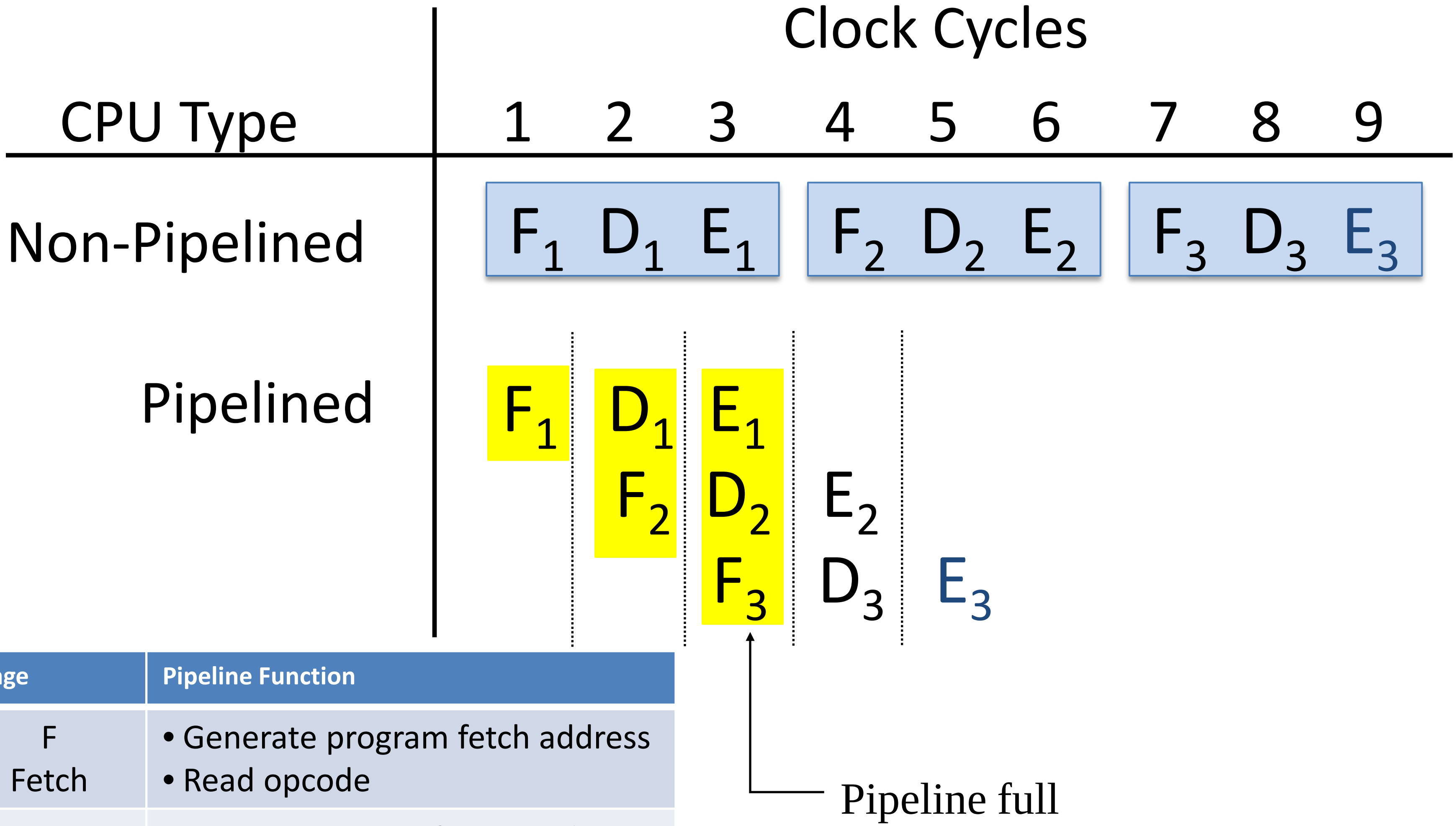


C62x: Dual 32-Bit Load/Store
C67x: Dual 64-Bit Load / 32-Bit Store
C64x, C674x, C66x: Dual 64-Bit Load/Store

Pipeline Concept

1. CorePac Architecture
2. Single Instruction Multiple Data (SIMD)
3. Memory Access
4. Pipeline Concept

Non-Pipelined vs. Pipelined CPU

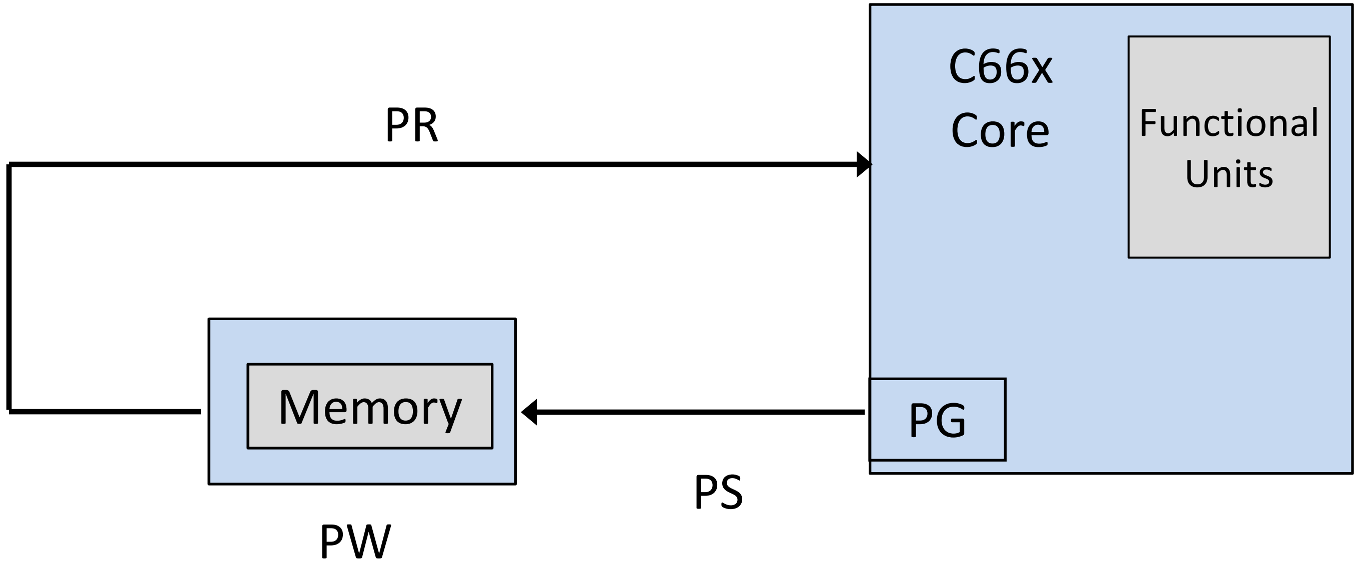


Stage	Pipeline Function
F Fetch	• Generate program fetch address • Read opcode
D Decode	• Route opcode to functional units • Decode instructions
E Execute	Execute instructions

Now look at the C66x pipeline.

Program Fetch Phases

Phase	Description
PG	Generate fetch address
PS	Send address to memory
PW	Wait for data ready
PR	Read opcode



Pipeline Phases - Review

Program Fetch

Decode

Execute

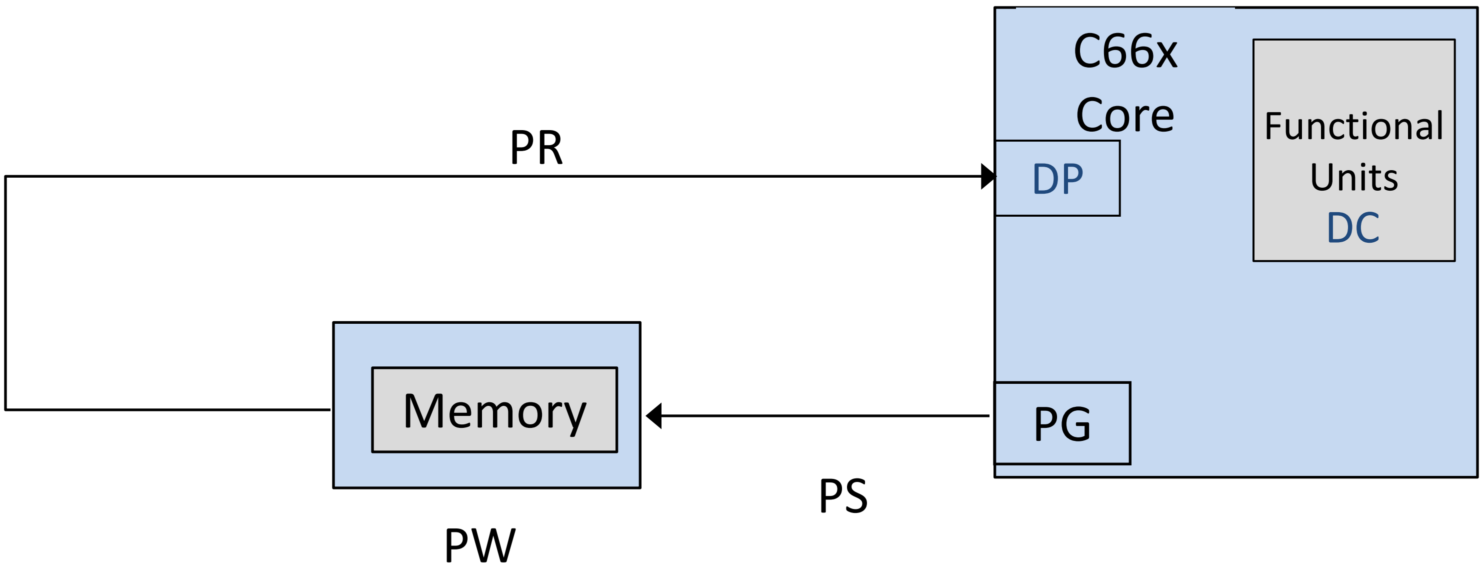
PG	PS	PW	PR	D	E					
	PG	PS	PW	PR	D	E				
		PG	PS	PW	PR	D	E			
			PG	PS	PW	PR	D	E		
				PG	PS	PW	PR	D	E	
					PG	PS	PW	PR	D	E

- Single-cycle performance is not affected by adding three program fetch phases.
- That is, there is still an execute every cycle.

How about decode? Is it only one cycle?

Decode Phases

Decode Phase	Description
DP	Intelligently routes instruction to functional unit (dispatch)
DC	Instruction decoded at functional unit (decode)



Pipeline Phases

Program Fetch				Decode		Execute						
PG	PS	PW	PR	DP	DC	E1						
	PG	PS	PW	PR	DP	DC	E1					
		PG	PS	PW	PR	DP	DC	E1				
			PG	PS	PW	PR	DP	DC	E1			
				PG	PS	PW	PR	DP	DC	E1		
					PG	PS	PW	PR	DP	DC	E1	
						PG	PS	PW	PR	DP	DC	E1

Pipeline Full

How many cycles does it take to execute an instruction?

Instruction Delays

All C66x instructions require only one cycle to execute, but some results are delayed.

Description	Instruction Example	Delay
Single Cycle	All instructions except	0
Integer multiplication and new floating point	MPY, FMPYSP	1
Legacy floating point multiplication	MPYSP	2
Load	LDW	4
Branch	B	5

Software Pipeline Example

Dot product; A typical DSP MAC operation.

```
   LDH
|| LDH
   MPY
   ADD
```

How many cycles would it take to perform this loop five times? (Disregard delay slots).

__________________ cycles

Software Pipeline Example

Dot product; A typical DSP MAC operation.

```
    LDH
||  LDH
    MPY
    ADD
```

How many cycles would it take to perform this loop five times? (Disregard delay slots).

5 x 3 = 15 cycles

Non-Pipelined Code

Cycle								
1	ldh	ldh	.M1	.M2	.L1	.L2	.S1	.S2
2			mpy					
3					add			
4	ldh	ldh						
5			mpy					
6					add			
7	ldh	ldh						
8			mpy					
9					add			

Pipelining Code

Cycle								
1	ldh	ldh	.M1	.M2	.L1	.L2	.S1	.S2
2	ldh	ldh	mpy					
3	ldh	ldh	mpy		add			
4	ldh	ldh	mpy		add			
5	ldh	ldh	mpy		add			
6	No LDHs?		mpy		add			
7					add			

Pipelining these instructions took 1/2 the cycles!

Software Pipeline Support

- The compiler is smart enough to schedule instructions efficiently.
- DSP algorithms are typically loop intensive.
- Generally speaking, servicing of interrupts is not allowed in the middle of the loop because fixed timing is essential.
- The C66x hardware SPLOOP enables servicing of interrupts in the middle of loops.

NOTE: For more information on SPLOOP, refer to Chapter 8 of the C66x CPU and Instruction Set Reference Guide.

For More Information

- For more information, refer to the C66x CPU and Instruction Set Reference Guide.
- For questions regarding topics covered in this training, visit the support forums at the TI E2E Community website.