D. Richard Brown III
Associate Professor
Worcester Polytechnic Institute
Electrical and Computer Engineering Department
drb@ece.wpi.edu

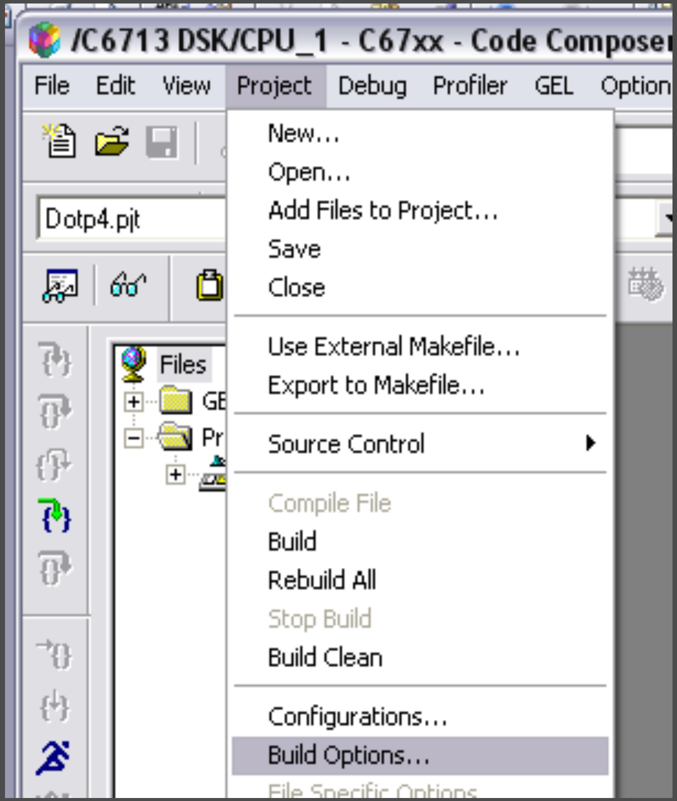
24-November-2008

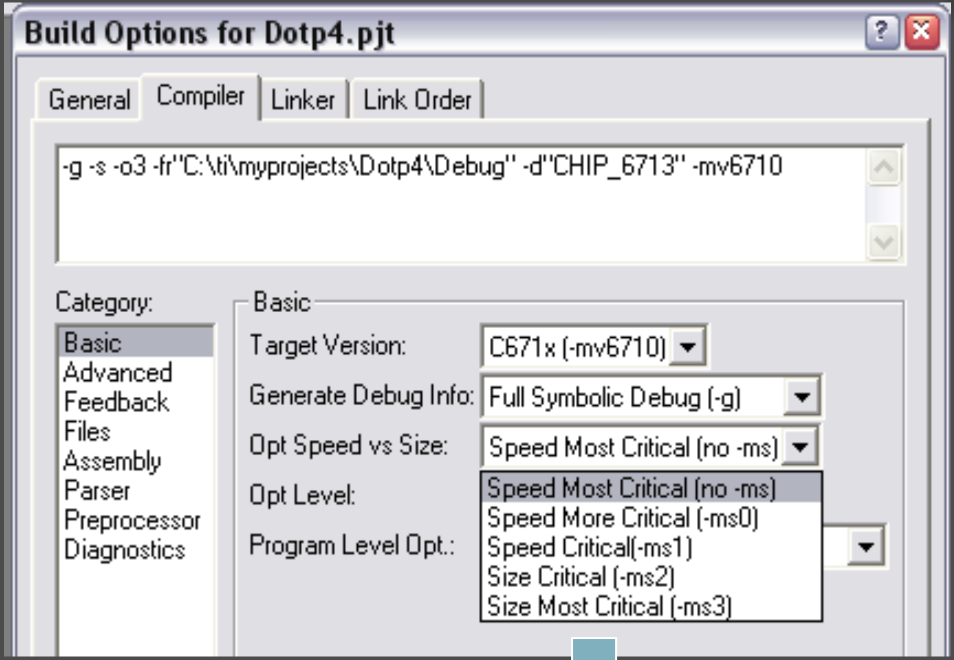
ECE4703 Real-Time DSP TMS320C6713 Architecture Overview and Assembly Language Programming

Outline

- Letting CCS optimize your code for you
- Not good enough? Writing your own assembly language functions for the C6x
- C6x architecture specifics:
  - Registers
  - Functional units
  - Pipelining
  - Fetch/execute packets

Optimizing Compiler

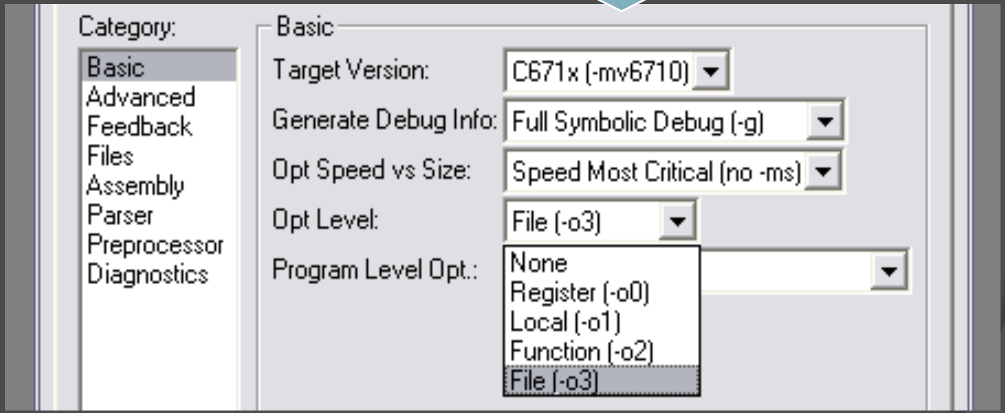



Build Options for Dotp4.pjt

```
-g -s -o3 -fr"C:\ti\myprojects\Dotp4\Debug" -d"CHIP_6713" -mv6710
```

Target Version: C671x (-mv6710)
Generate Debug Info: Full Symbolic Debug (-g)
Opt Speed vs Size: Speed Most Critical (no -ms)
Opt Level:
Program Level Opt.:

Speed Most Critical (no -ms)
Speed More Critical (-ms0)
Speed Critical(-ms1)
Size Critical (-ms2)
Size Most Critical (-ms3)



Target Version: C671x (-mv6710)
Generate Debug Info: Full Symbolic Debug (-g)
Opt Speed vs Size: Speed Most Critical (no -ms)
Opt Level: File (-o3)
Program Level Opt.:

Typical speed gains of 2x to 5x with compiler optimization.

Assembly Language Programming on the TMS320C6713

- To achieve the best possible performance, sometimes you have to take matters into your own hands...
- Three options:
  1. Linear assembly (.sa)
     - Compromise between effort and efficiency
     - Typically more efficient than C
     - Assembler takes care of details like assigning "functional units", registers, and parallelizing instructions
  2. ASM statement in C code (.c)
     - asm("assembly code")
  3. C-callable assembly function (.asm)
     - Full control of assigning functional units, registers, parallelization, and pipeline optimization

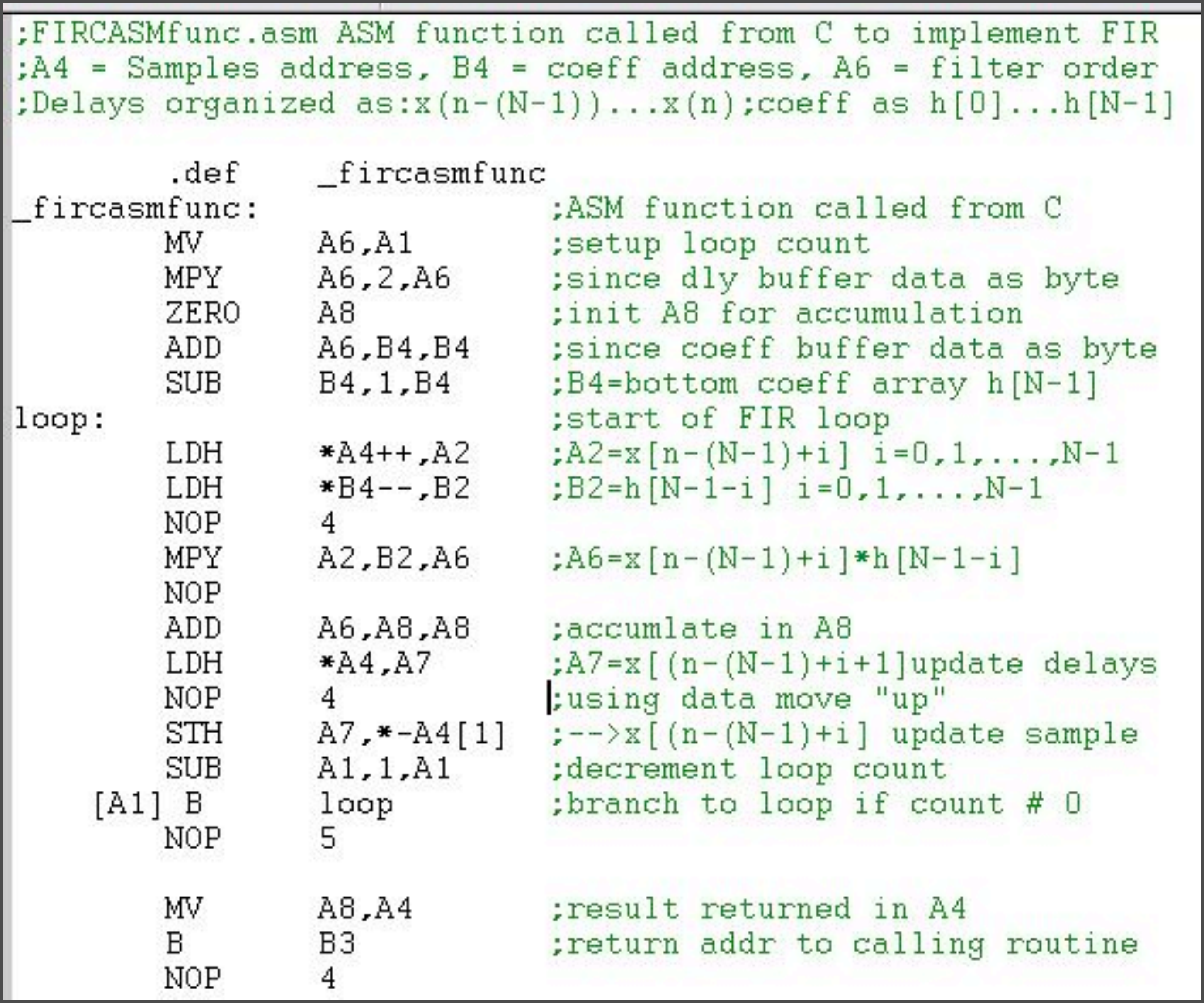
C-Callable Assembly Language Functions

- Basic concepts:
  - Arguments are passed in via registers A4, B4, A6, B6, … in that order. All registers are 32-bit.
  - Result returned in A4 also.
  - Return address of calling code (program counter) is in B3. Don't overwrite B3!
  - Naming conventions:
    - In C code: label
    - In ASM code: _label (note the leading underbar)
  - Accessing global variables in ASM:
    - .ref _variablename
  - A function prototype must also be included in your C code.

Skeleton C-Callable ASM Function

```
; header comments
; passed in parameters in registers A4, B4, A6, ... in that order

            .def _myfunc                ; allow calls from external
ACONSTANT   .equ 100                    ; declare constants
            .ref _aglobalvariable       ; refer to a global variable

_myfunc:    NOP                         ; instructions go here
            B            B3             ; return (branch to addr B3)
                                        ; function output will be in A4
            NOP          5              ; pipeline flush

            .end
```

Example C-Callable Assembly Language Program (Chassaing)
int fircasmfunc(short x[], short h[], int N)

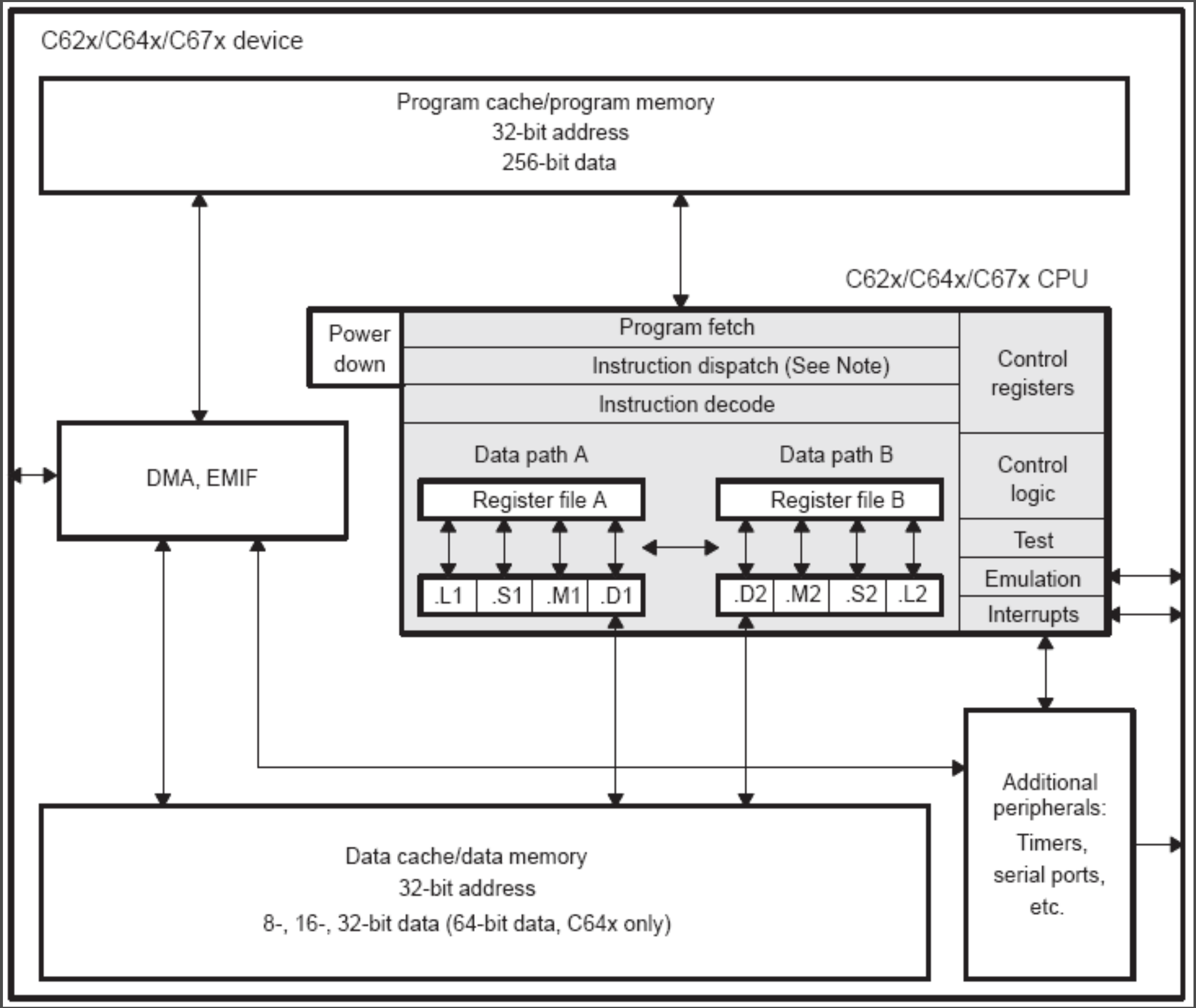


```
;FIRCASMfunc.asm ASM function called from C to implement FIR
;A4 = Samples address, B4 = coeff address, A6 = filter order
;Delays organized as:x(n-(N-1))...x(n);coeff as h[0]...h[N-1]

        .def    _fircasmfunc
_fircasmfunc:                    ;ASM function called from C
        MV      A6,A1            ;setup loop count
        MPY     A6,2,A6          ;since dly buffer data as byte
        ZERO    A8               ;init A8 for accumulation
        ADD     A6,B4,B4         ;since coeff buffer data as byte
        SUB     B4,1,B4          ;B4=bottom coeff array h[N-1]
loop:                            ;start of FIR loop
        LDH     *A4++,A2         ;A2=x[n-(N-1)+i] i=0,1,...,N-1
        LDH     *B4--,B2         ;B2=h[N-1-i] i=0,1,...,N-1
        NOP     4
        MPY     A2,B2,A6         ;A6=x[n-(N-1)+i]*h[N-1-i]
        NOP
        ADD     A6,A8,A8         ;accumlate in A8
        LDH     *A4,A7           ;A7=x[(n-(N-1)+i+1]update delays
        NOP     4                ;using data move "up"
        STH     A7,*-A4[1]       ;-->x[(n-(N-1)+i] update sample
        SUB     A1,1,A1          ;decrement loop count
    [A1] B      loop             ;branch to loop if count # 0
        NOP     5

        MV      A8,A4            ;result returned in A4
        B       B3               ;return addr to calling routine
        NOP     4
```

Writing Efficient Assembly Language Programs for the C6x

- Need to become familiar with:
  - Specific architecture, capabilities, and limitations of the C6x
    - Registers
    - Functional units
    - Pipeline
    - Parallelization
    - …
  - Instruction set
    - Different commands for single precision floating point, double precision floating point, and integer math

TMS320C67x Block Diagram



One instruction is 32 bits. Program bus is 256 bits wide.

⇨Can execute up to 8 instructions per clock cycle (225MHz->4.4ns clock cycle).

8 independent functional units:
- 2 multipliers
- 6 ALUs

Code is efficient if all 8 functional units are always busy.

Register files each have 16 general purpose registers, each 32-bits wide (A0-A15, B0-B15).

C6713 Data Paths and Functional Units

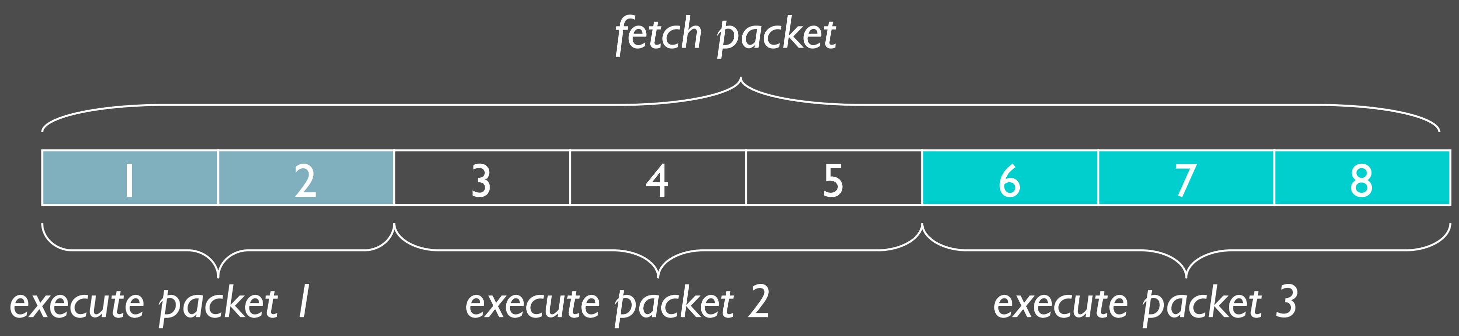
- Two data paths (A & B)
- Data path A
  - Multiply operations (.M1)
  - Logical and arithmetic operations (.L1)
  - Branch, bit manipulation, and arithmetic operations (.S1)
  - Loading/storing and arithmetic operations (.D1)
- Data path B
  - Multiply operations (.M2)
  - Logical and arithmetic operations (.L2)
  - Branch, bit manipulation, and arithmetic operations (.S2)
  - Loading/storing and arithmetic operations (.D2)
- All data (not program) transfers go through .D1 and .D2

Fetch & Execute Packets

- C6713 fetches 8 instructions at a time (256 bits)
- Definition: "Fetch packet" is a group of 8 instructions fetched at once.
- Coincidentally, C6713 has 8 functional units.
  - Ideally, all 8 instructions are executed in parallel.
- Often this isn't possible, e.g.:
  - 3 multiplies (only two .M functional units)
  - Results of instruction 3 needed by instruction 4 (must wait for 3 to complete)

Execute Packets

- Definition: "Execute Packet" is a group of (8 or less) consecutive instructions in one fetch packet that can be executed in parallel.



- C compiler provides a flag to indicate which instructions should be run in parallel.
- You have to do this manually in Assembly using the double-pipe symbol "||".

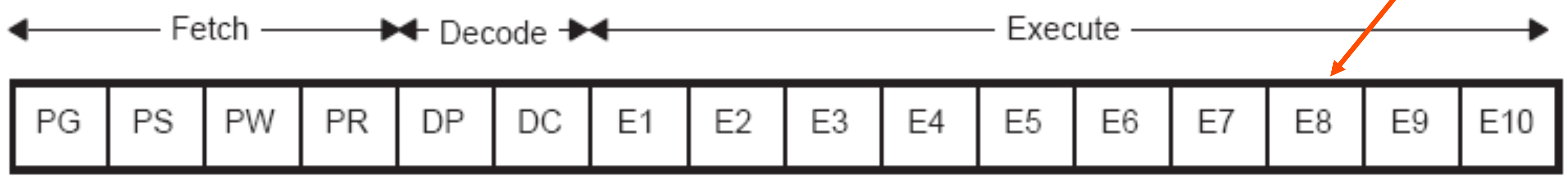
C6713 Instruction Pipeline Overview

All instructions flow through the following steps:

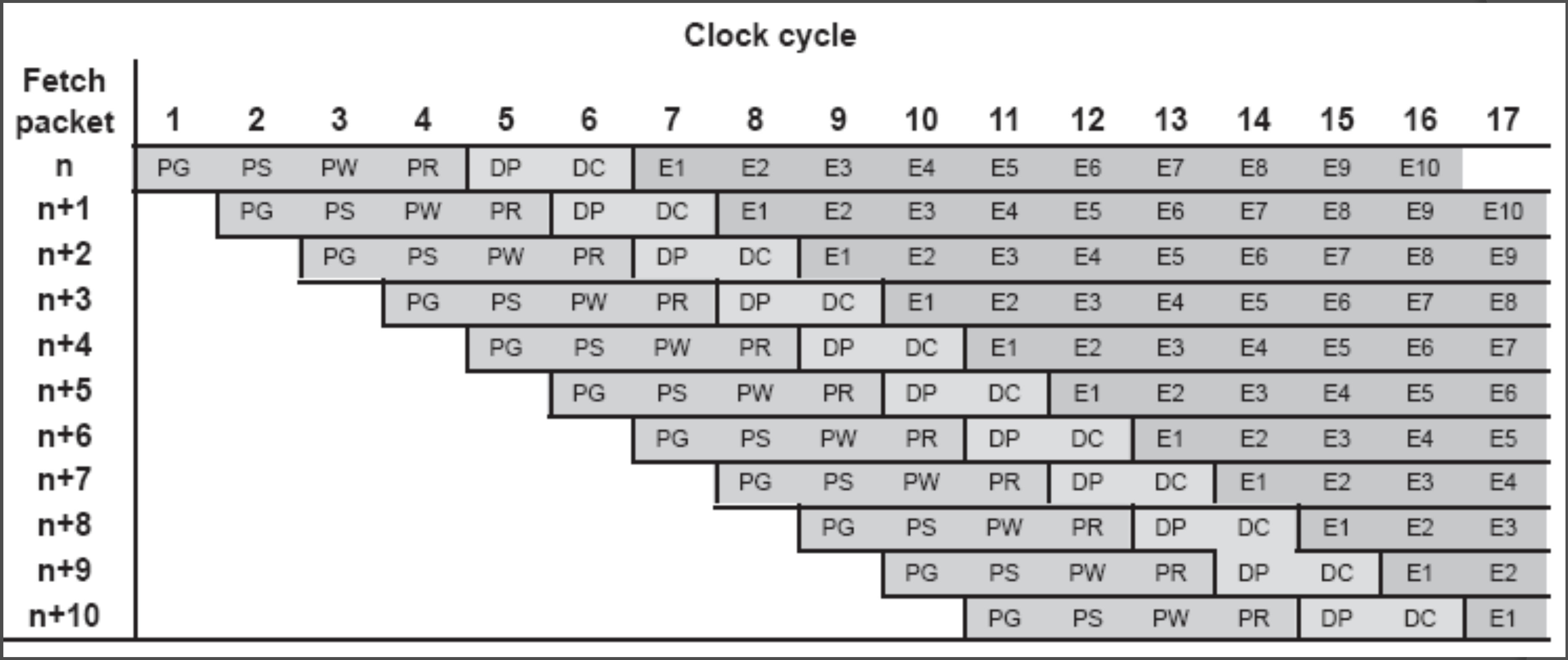
1. Fetch
   a) PG: Program address Generate
   b) PS: Program address Send
   c) PW: Program address ready Wait
   d) PR: Program fetch packet Receive
2. Decode
   a) DP: Instruction DisPatch
   b) DC: Instruction DeCode
3. Execute
   a) 10 phases labeled E1-E10
   b) Fixed point processors have only 5 phases (E1-E5)

each step = 1 clock cycle

Figure 7–5. Floating-Point Pipeline Phases



Pipelining: Ideal Operation



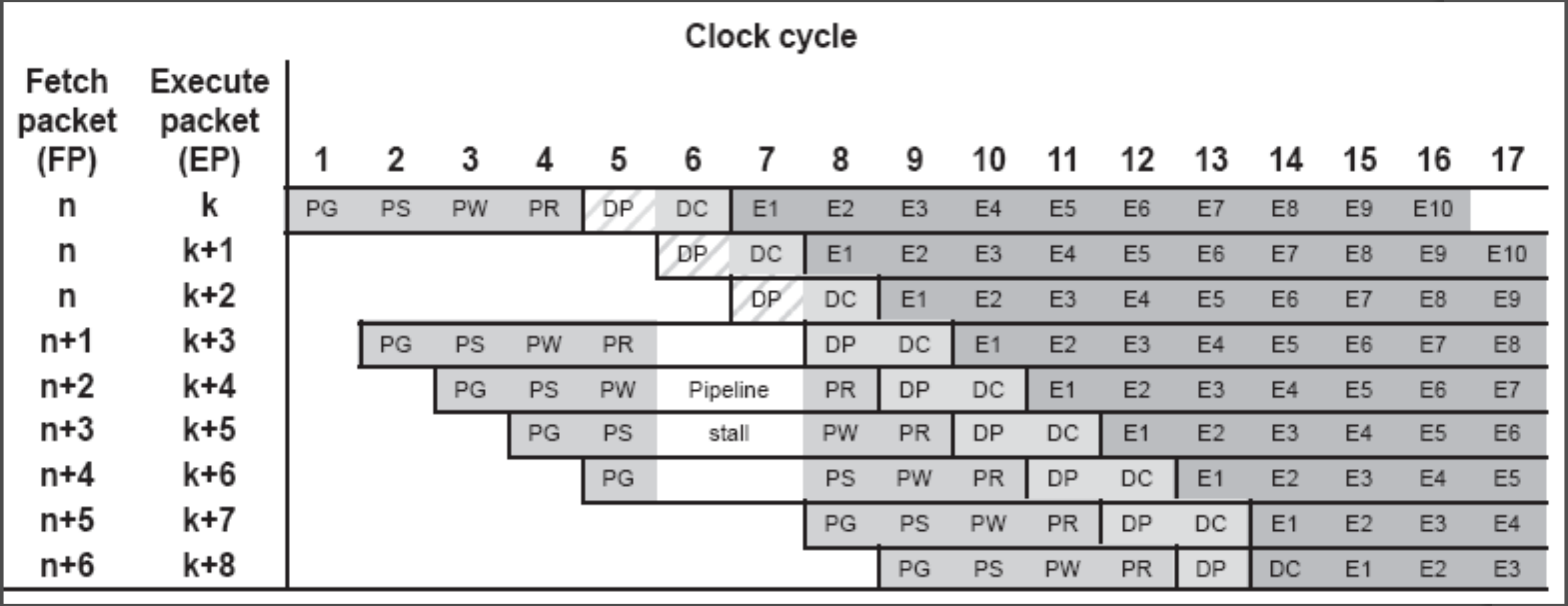
Fetch packet	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
n	PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	
n+1		PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
n+2			PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	E6	E7	E8	E9
n+3				PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	E6	E7	E8
n+4					PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	E6	E7
n+5						PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	E6
n+6							PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5
n+7								PG	PS	PW	PR	DP	DC	E1	E2	E3	E4
n+8									PG	PS	PW	PR	DP	DC	E1	E2	E3
n+9										PG	PS	PW	PR	DP	DC	E1	E2
n+10											PG	PS	PW	PR	DP	DC	E1

Clock cycle

Remarks:

- At clock cycle 11, the pipeline is "full"
- There are no holes ("bubbles") in the pipeline in this example

Pipelining: "Actual" Operation



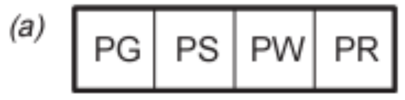
		Clock cycle																
Fetch packet (FP)	Execute packet (EP)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
n	k	PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	
n	k+1						DP	DC	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
n	k+2							DP	DC	E1	E2	E3	E4	E5	E6	E7	E8	E9
n+1	k+3		PG	PS	PW	PR			DP	DC	E1	E2	E3	E4	E5	E6	E7	E8
n+2	k+4			PG	PS	PW	Pipeline		PR	DP	DC	E1	E2	E3	E4	E5	E6	E7
n+3	k+5				PG	PS	stall		PW	PR	DP	DC	E1	E2	E3	E4	E5	E6
n+4	k+6					PG			PS	PW	PR	DP	DC	E1	E2	E3	E4	E5
n+5	k+7								PG	PS	PW	PR	DP	DC	E1	E2	E3	E4
n+6	k+8									PG	PS	PW	PR	DP	DC	E1	E2	E3

Remarks:

- Fetch packet n has 3 execution packets
- All subsequent fetch packets have 1 execution packet
- Notice the holes/bubbles in the pipeline caused by lack of parallelization

Fetch Phases of C6713 Pipeline

Figure 7–2. Fetch Phases of the Pipeline

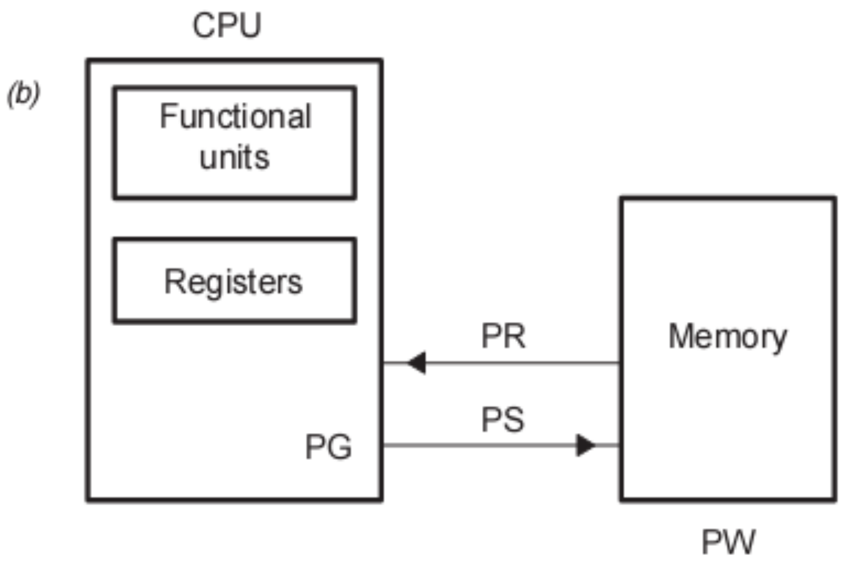


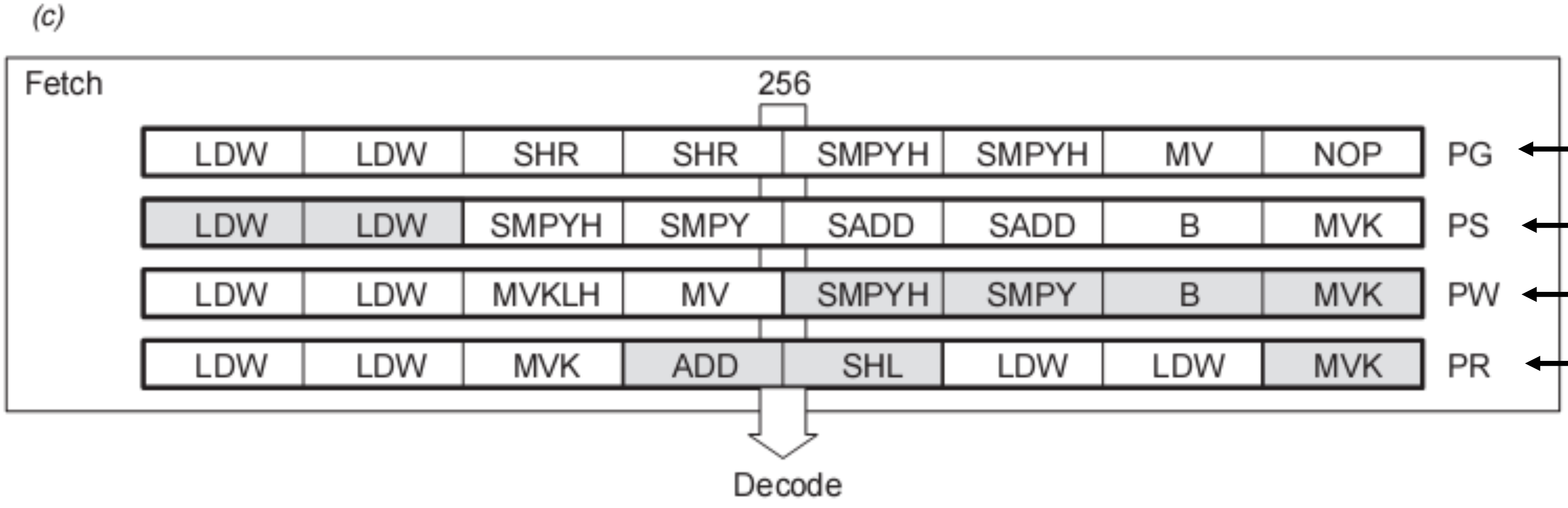
PG: Program Address Generate

PS: Program Address Send

PW: Program Address Ready Wait

PR: Program Fetch Packet Receive




(c)
Fetch
256
LDW LDW SHR SHR SMPYH SMPYH MV NOP PG
LDW LDW SMPYH SMPY SADD SADD B MVK PS
LDW LDW MVKLH MV SMPYH SMPY B MVK PW
LDW LDW MVK ADD SHL LDW LDW MVK PR
Decode

PG ← 1 EP

PS ← 2 EP

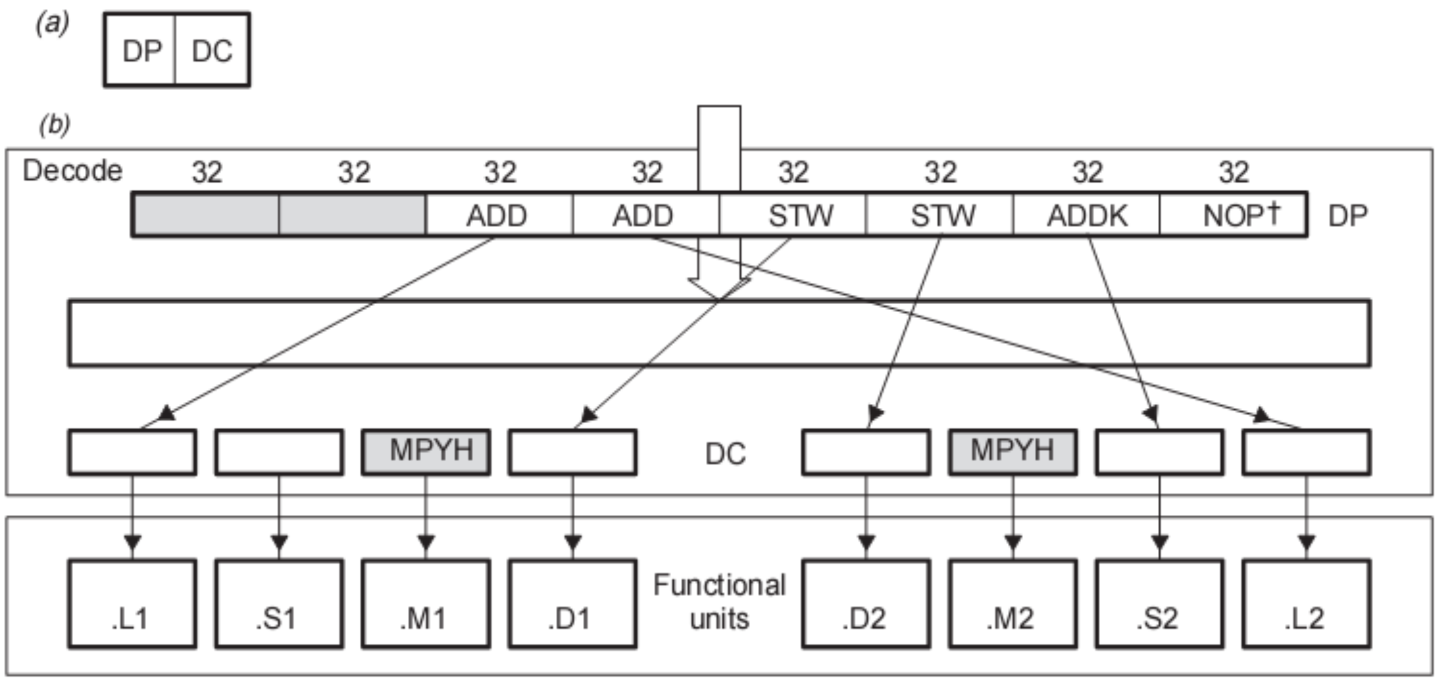
PW ← 2 EP

PR ← 4 EP

Decode Phases of C6713 Pipeline

- DP (instruction dispatch) phase
  - Fetch packets (FPs) are split into execute packets (EPs)
  - Instructions in an EP are assigned to appropriate functional units for decoding
- DC (instruction decode) phase: convert instruction to microcode for appropriate functional unit

Figure 7–3. Decode Phases of the Pipeline



† NOP is not dispatched to a functional unit.

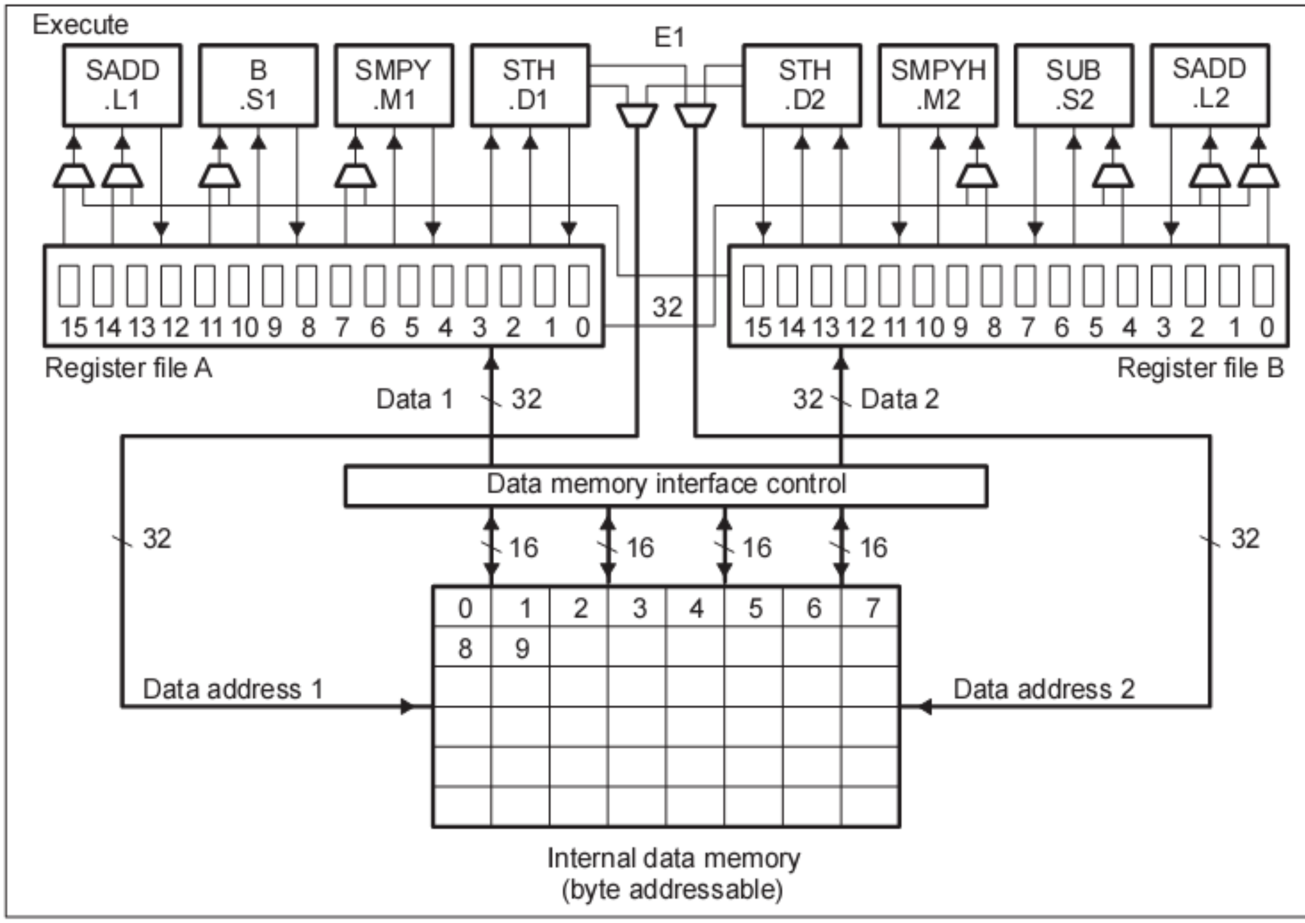
Execute Phases of C6713 Pipeline

Figure 7–4. Execute Phases of the Pipeline and Functional Block Diagram of the TMS320C67x

(a)

E1	E2	E3	E4	E5	E6	E7	E8	E9	E10

(b)



Execute Phases of C6713 Pipeline

- C67x has 10 execute phases (floating point)

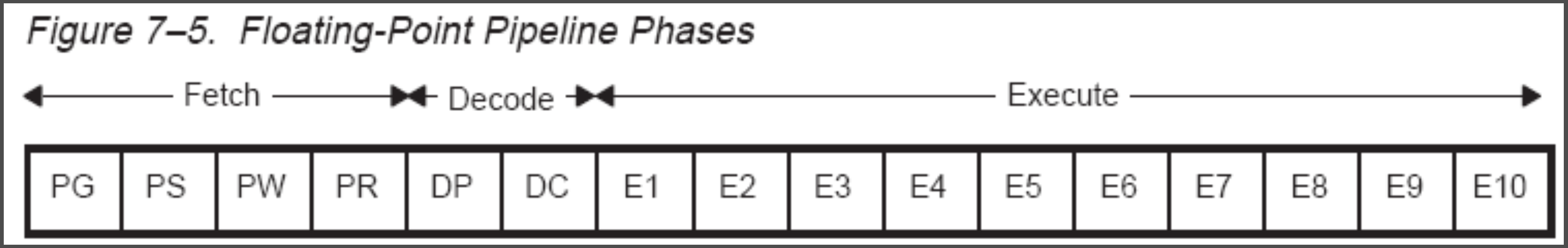


Figure 7–5. Floating-Point Pipeline Phases

- C62x/C64x have 5 execute phases (fixed point)
- Different types of instructions require different numbers of execute phases to complete
  - Anywhere between 1 and all 10 phases
  - Most instruction tie up their functional unit for only one phase (E1)

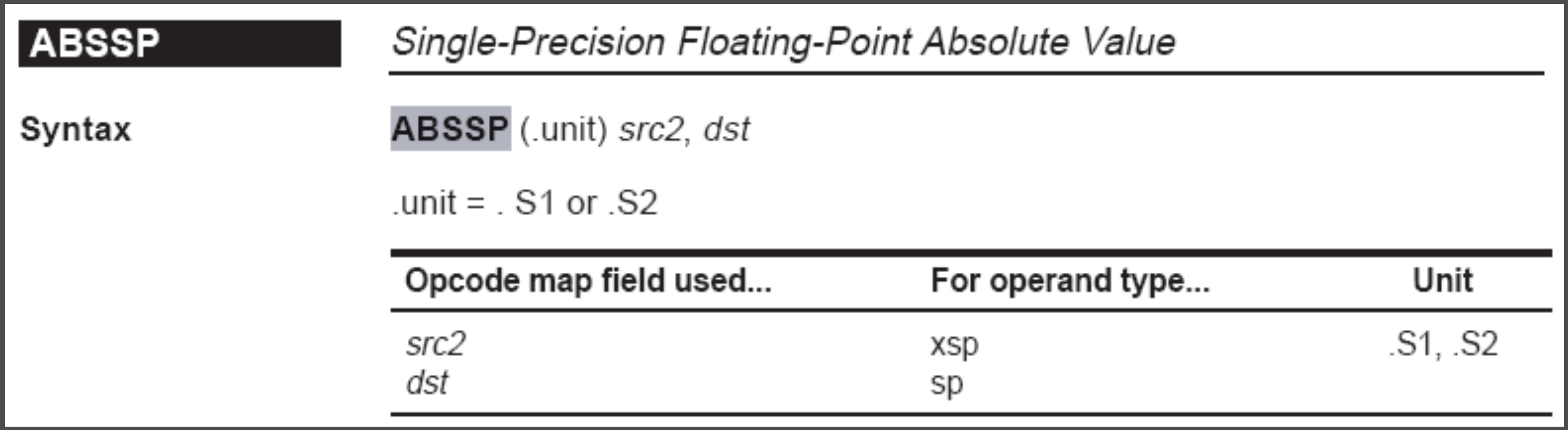
Execute Stage: Delay Slots

- How long must we wait for the result of an instruction?
  - Most instructions' results are available at the end of E1 (called "*single-cycle*" instructions)
    - Examples:
      - ABSSP (single precision absolute value)
      - RCPSP (single precision reciprocal approximation)
  - Some instructions take more time to produce results
    - Examples:
      - MPYSP (single precision multiply): Results available at the end of E4 (3 delays slots)
      - ADDSP (single precision addition): Results available at the end of E4 (3 delay slots)

Execute Stage: Functional Latency

- How long must we wait for the functional unit to be free?
  - Most instructions tie up the functional unit for only one pipeline stage (E1)
    - Examples:
      - All single-cycle instructions
      - Most multicycle instructions, including, for example, ADDSP (single precision addition)
  - Some instructions tie up the execution unit for more than one pipeline stage
    - Examples:
      - MPYDP (double precision multiply): .M execution unit is tied up for 4 pipeline stages (E1-E4). Can't use this functional unit until E4 completes.

Execution Stage Examples (1)

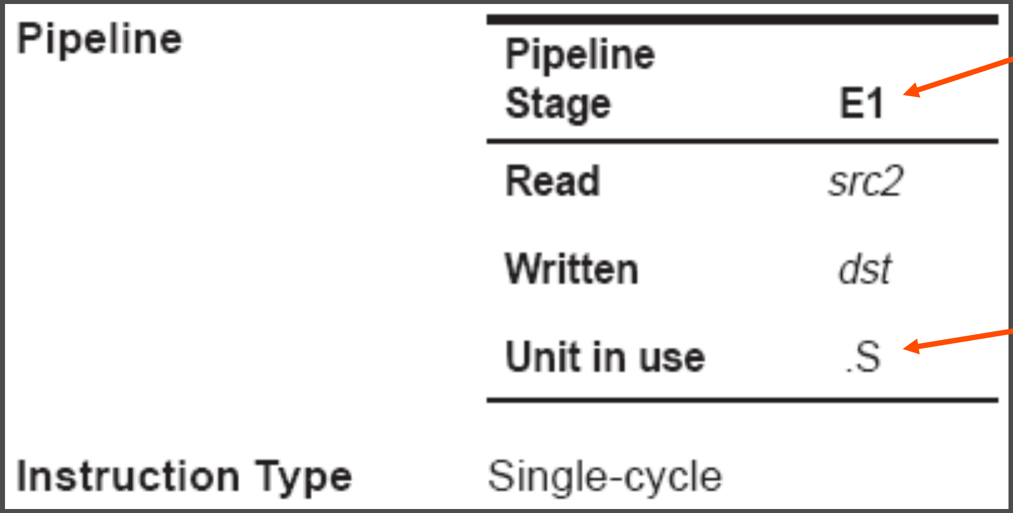


ABSSP *Single-Precision Floating-Point Absolute Value*

Syntax **ABSSP** (.unit) *src2*, *dst*

.unit = . S1 or .S2

Opcode map field used...	For operand type...	Unit
src2 *dst*	xsp sp	.S1, .S2



Pipeline

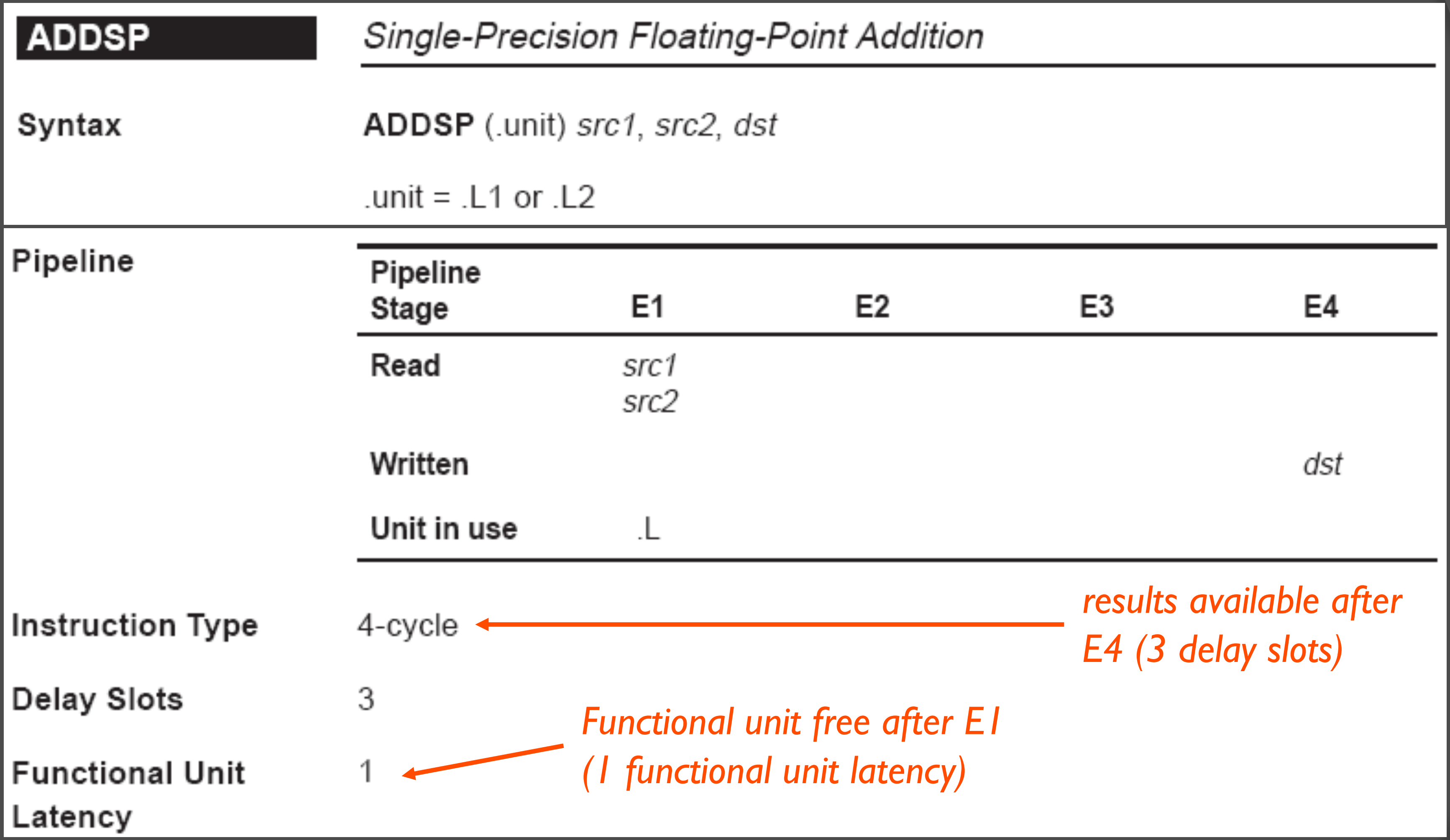
Pipeline Stage	E1
Read	*src2*
Written	*dst*
Unit in use	.S

Instruction Type Single-cycle

results available after E1 (zero delay slots)

Functional unit free after E1 (1 functional unit latency)

Execution Stage Examples (2)



ADDSP *Single-Precision Floating-Point Addition*

Syntax **ADDSP** (.unit) *src1*, *src2*, *dst*

.unit = .L1 or .L2

Pipeline

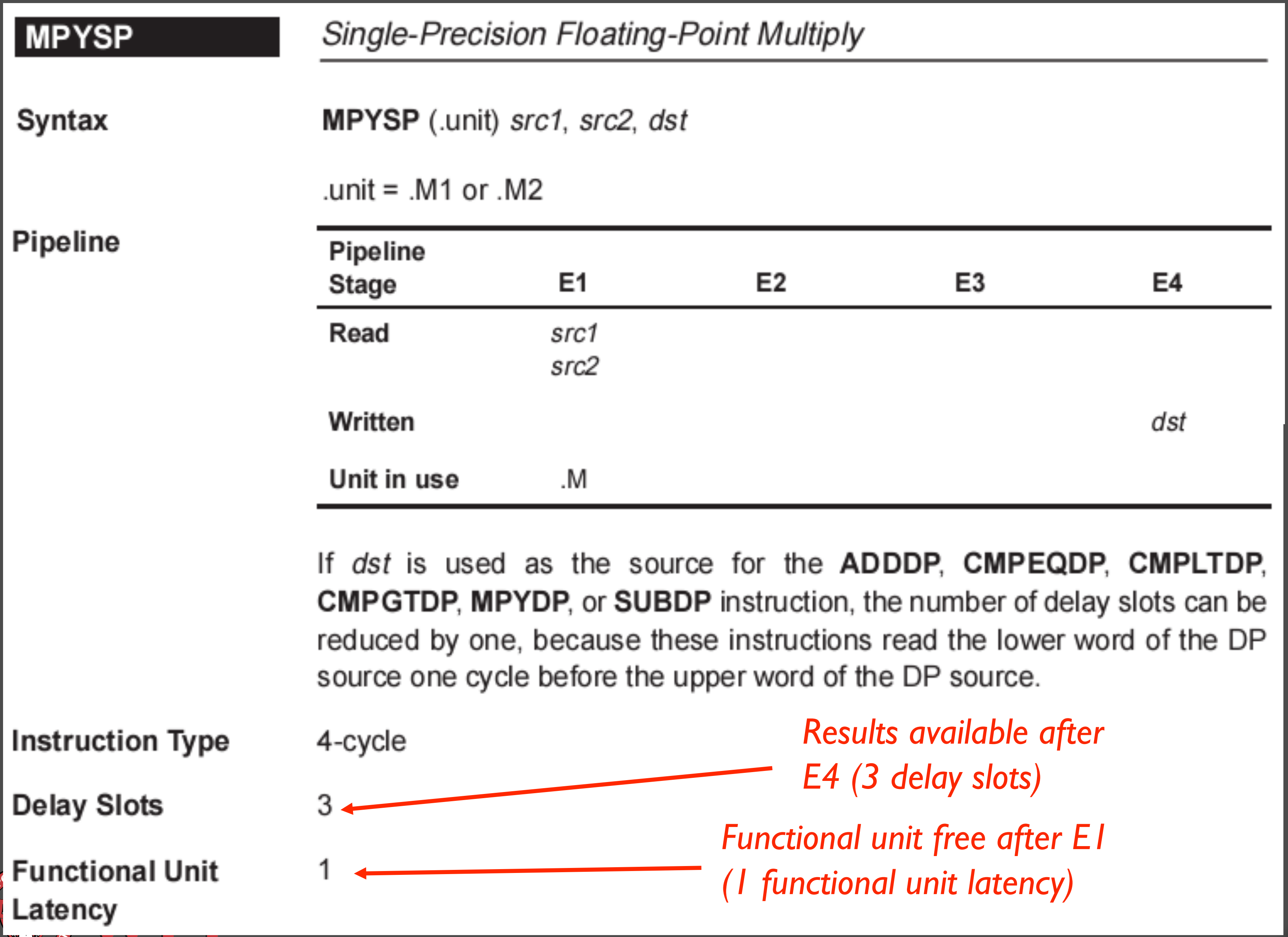
Pipeline Stage	**E1**	**E2**	**E3**	**E4**
Read	*src1* *src2*			
Written				*dst*
Unit in use	.L			

Instruction Type	4-cycle
Delay Slots	3
Functional Unit Latency	1

results available after E4 (3 delay slots)

Functional unit free after E1 (1 functional unit latency)

Execution Stage Examples (3)



MPYSP *Single-Precision Floating-Point Multiply*

Syntax **MPYSP** (.unit) *src1*, *src2*, *dst*

.unit = .M1 or .M2

Pipeline

Pipeline Stage	**E1**	**E2**	**E3**	**E4**
Read	*src1* *src2*			
Written				*dst*
Unit in use	.M			

If *dst* is used as the source for the **ADDDP**, **CMPEQDP**, **CMPLTDP**, **CMPGTDP**, **MPYDP**, or **SUBDP** instruction, the number of delay slots can be reduced by one, because these instructions read the lower word of the DP source one cycle before the upper word of the DP source.

Instruction Type 4-cycle

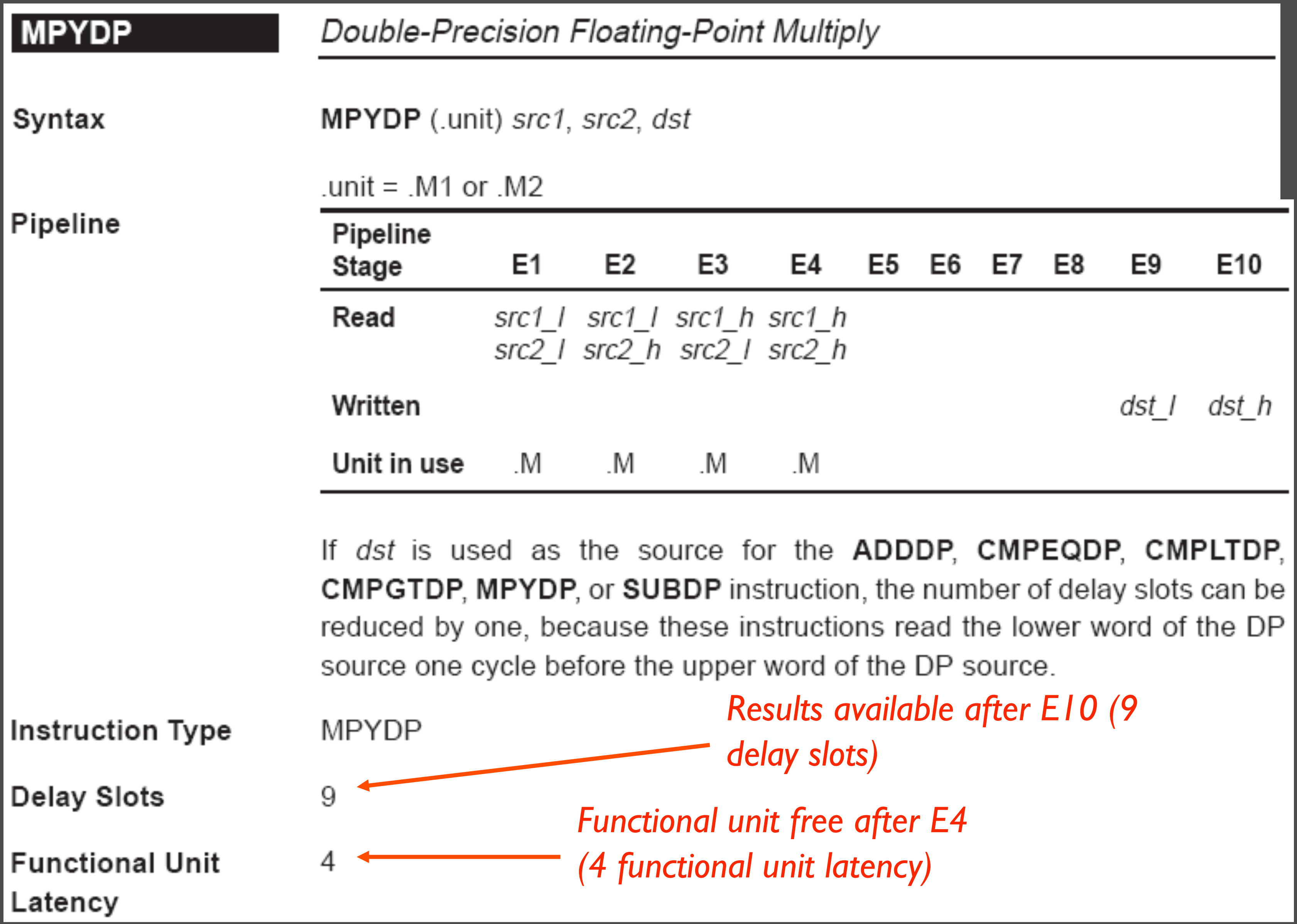
Delay Slots 3

Functional Unit Latency 1

Results available after E4 (3 delay slots)

Functional unit free after E1 (1 functional unit latency)

Execution Stage Examples (4)



MPYDP *Double-Precision Floating-Point Multiply*

Syntax **MPYDP** (.unit) *src1*, *src2*, *dst*

.unit = .M1 or .M2

Pipeline

Pipeline Stage	**E1**	**E2**	**E3**	**E4**	**E5**	**E6**	**E7**	**E8**	**E9**	**E10**
Read	*src1_l* *src2_l*	*src1_l* *src2_h*	*src1_h* *src2_l*	*src1_h* *src2_h*						
Written									*dst_l*	*dst_h*
Unit in use	.M	.M	.M	.M						

If *dst* is used as the source for the **ADDDP**, **CMPEQDP**, **CMPLTDP**, **CMPGTDP**, **MPYDP**, or **SUBDP** instruction, the number of delay slots can be reduced by one, because these instructions read the lower word of the DP source one cycle before the upper word of the DP source.

Instruction Type MPYDP

Delay Slots 9

Functional Unit Latency 4

Results available after E10 (9 delay slots)

Functional unit free after E4 (4 functional unit latency)

Delay Slots & Functional Latency

- IMPORTANT: Delay slots are not the same as functional unit latency
- Example:

```
MPYSP .M1 A1,A2,A3          ;A3 = A1 x A2
MPYSP .M1 A4,A5,A6          ;A6 = A4 x A5
MPYSP .M1 A7,A8,A9          ;A9 = A6 x A7
MPYSP .M1 A10,A11,A12       ;A12 = A10 x A11
```

- Is this code ok?

Delay Slots & Functional Latency

- What about this code?

```
MPYSP .M1 A1,A2,A3        ;A3 = A1 x A2
MPYSP .M1 A3,A4,A5        ;A5 = A3 x A4
```

Delay Slots & Functional Latency

- You are probably going to get strange results here because the result in A3 is not available until E4 completes for the first MPYSP instruction
- "Data hazard" due to the delay slots in MPYSP
- How to "fix" the last example

```
MPYSP .M1 A1,A2,A3      ;A3 = A1 x A2
NOP       3             ; insert 3 delay slots
                         ; results of first multiply now in A3
MPYSP .M1 A3,A4,A5      ;A5 = A3 x A4
```

Delay Slots & Functional Latency

- What about this code?

```
MPYDP .M1 A1:A0, A3:A2, A5:A4
MPYDP .M1 A7:A6, A9:A8, A11:A10
```

Delay Slots & Functional Latency

- This won't work because the functional unit M1 is tied up for 4 clock cycles (E1-E4) by MPYDP
- "Resource conflict" due to the functional latency in MPYDP
- How to fix it:

```
MPYDP .M1 A1:A0, A3:A2, A5:A4
NOP           3        ; 3 NOPs for func latency
MPYDP .M1 A7:A6, A9:A8, A11:A10
```

Delay Slots & Functional Latency

- What about this code?

```
MPYDP .M1 A1:A0,A3:A2,A5:A4
MPYDP .M1 A5:A4,A8:A7,A11:A10
```

Delay Slots & Functional Latency

- Two problems now!
  - Resource conflict for .M1 unit (E2-E4)
  - Data hazard for result in A5:A4 (E2-E10)
- The "fix":

```
MPYDP .M1 A1:A0, A3:A2, A5:A4
NOP                  9
MPYDP .M1 A5:A4, A8:A7, A11:A10
```

- Note: Could use M1 after E4, but A5:A4 not available until after E10.

Functional Latency & Delay Slots

- Functional Latency: How long must we wait for the functional unit to be free?
- Delay Slots: How long must we wait for the result of a calculation to be available?
- General remarks:
  - Functional unit latency <= Delay slots
  - Strange results will occur in ASM code if you don't pay attention to delay slots and functional unit latency
  - All problems can be resolved by "waiting" with NOPs
  - Efficient ASM code tries to keep functional units busy all of the time.
  - Efficient code is hard to write (and follow).

Additional Constraints: Data Cross-Paths

- TMS320C6x core has A side and B side
  - A side: M1, S1, L1, D1, and register file A0-A15
  - B side: M2, S2, L2, D2, and register file B0-B15

- Cross path instruction examples:
  - MPYSP .M1x A2, B2, A4 ; cross path brings B2 to M1
  - MPYSP .M2x A2, B2, B4 ; cross path brings A2 to M2

- Constraint: Only two cross-paths are available per cycle: A→2 and B→1.
  - Note: Can't have two A→2 or two B→1 cross paths in the same cycle.

Additional Constraints

- Memory constraints
  - Two memory accesses can be performed in one cycle if they don't access the same bank of memory
  - See TMS320C6000 Programmer's Guide

- Load/Store constraints
  - Address register must agree with .D unit, e.g.:
    - LDW .D1 *A1,A2 ; valid because A1 and D1 agree
  - Parallel loads and stores must use different register files
  - See TMS320C6000 Programmer's Guide

Suggested Reading

- Reference material (on course web page)
  - TMS320C6000 CPU Instruction Set and Reference Guide
  - TMS320C6000 Programmer’s Guide
- Kehtarnavaz Chap 3
- Kehtarnavaz Chap 7
- Chassaing has some ASM example code in the myproject directory (see, for example, FIRcASM.pjt)