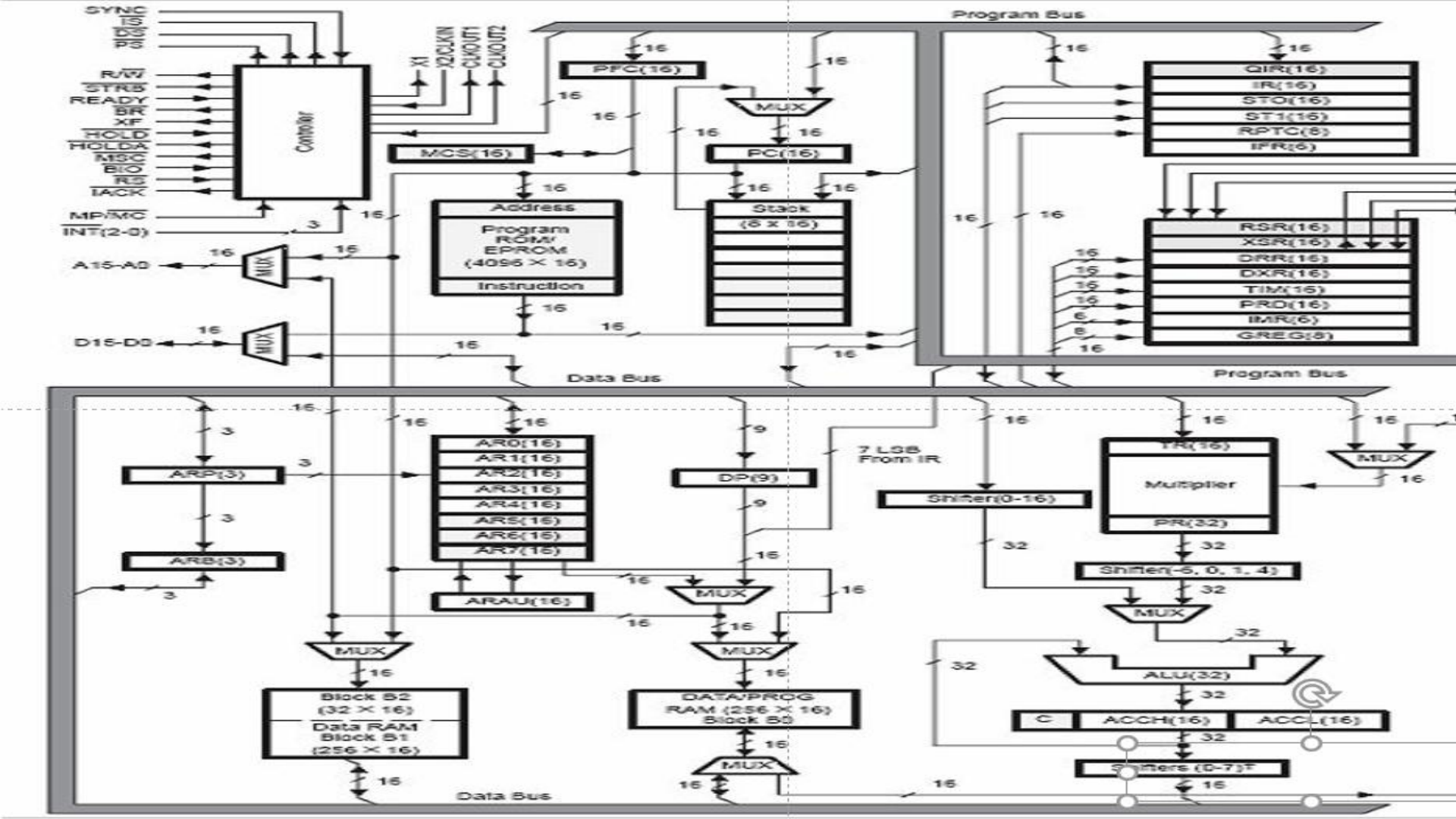
C5. C2x ASSEMBLY LANGUAGE

OUTLINE:

- Memory Addressing Modes
- Instructions set
- Examples
- Homework


SYNC
IS
DS
PS
R/W
STRB
READY
BR
XF
HOLD
HOLDA
MSC
BIO
RS
IACK
MP/MC
INT(2-0)
Controller
X1
X2/CLKIN
CLKOUT1
CLKOUT2
Program Bus
PFC(16)
MUX
PC(16)
MCS(16)
Address
Program ROM/ EPROM (4096 × 16)
Instruction
Stack (8 x 16)
A15-A0
D15-D0
MUX
Data Bus
QIR(16)
IR(16)
STO(16)
ST1(16)
RPTC(8)
IFR(6)
RSR(16)
XSR(16)
DRR(16)
DXR(16)
TIM(16)
PRD(16)
IMR(6)
GREG(8)
Program Bus
ARP(3)
ARB(3)
AR0(16)
AR1(16)
AR2(16)
AR3(16)
AR4(16)
AR5(16)
AR6(16)
AR7(16)
ARAU(16)
DP(9)
7 LSB From IR
Shifter(0-16)
TR(16)
Multiplier
PR(32)
MUX
Shifter(-6, 0, 1, 4)
ALU(32)
C
ACCH(16)
ACCL(16)
Shifters (0-7)
Block B2 (32 × 16)
Data RAM Block B1 (256 × 16)
DATA/PROG RAM (256 × 16) Block B0
Data Bus

TMS320C25 — Block Architecture

Harvard · 16×16-bit Multiplier · Fixed-Point DSP · 100 ns Cycle

ON-CHIP MEMORY

- **Program ROM** 4K × 16-bit (MP/M̄C̄=0) 0x0000–0x0FFF
- **RAM Block B0** 256 × 16-bit Prog or Data (CNFP / CNFD)
- **RAM Block B1** 256 × 16-bit Data Memory 0x0300–0x03FF
- **RAM Block B2** 32 × 16-bit Data 0x0060
- **Memory-Mapped Registers** 0x0000–0x005F DRR DXR TIM PRD IMR

544 words total on-chip RAM (B0+B1+B2)

CPU CORE

CALU — Central Arithmetic Logic Unit

- **Scaling Shifter** 0–16-bit left shift SXM control
- **32-bit ALU** 2's complement Boolean ops OVM saturation
- **32-bit ACC** (High / Low) ACC register
- **P Reg** 32-bit Product
- **16 × 16-bit Multiplier** → 32-bit product Single instruction cycle
- **T Reg** 16-bit Mult.
- **P-Shift** PM(2) bits

ST0: ARP(3) OV OVM DP(9) INTM

ST1: ARB(3) CNF TC SXM C XF HM FSM FO TXM PM(2)

Auxiliary Registers (ARAU)

AR0 · AR1 · AR2 · AR3 · AR4 · AR5 · AR6 · AR7

ARP · ARB

Program Control

- **16-bit PC** Program Counter
- **8-Level HW Stack** (16-bit words)
- **PFC** Prefetch Ctr
- **MCS Reg** (Microcall Stack)
- **IR** (Instr)
- **QIR** (Queue)
- **Repeat Ctr**

PERIPHERALS & I/O

- **On-Chip Timer** 16-bit TIM counter · 16-bit PRD TINT = CLKOUT1 / (PRD+1)

Full-Duplex Serial Port

- **DRR** Data Recv Reg
- **DXR** Data Xmit Reg
- **RSR** Recv Shift
- **XSR** Xmit Shift

DR DX CLKR CLKX FSR FSX · 8/16-bit (FO) · FSM mode

- **Interrupt Logic** INT0 INT1 INT2 TINT RINT XINT TRAP RS̄ IMR — Interrupt Mask Register
- **I/O Ports** 16 Input + 16 Output channels (IN / OUT instructions)
- **XF (O)** GP output pin
- **BIO (I)** Branch input
- Clock: X1/X2·CLKIN · CLKOUT1=CLKIN/4 · CLKOUT2 Q-phase

BUSES

- 16-bit DATA ADDRESS BUS (DAB) — Data Memory Space
- 16-bit PROGRAM ADDRESS BUS (PAB) — Program Memory Space
- D15–D0 (Data) / A15–A0 (Addr)
- D̄S̄ P̄S̄ ĪS̄ R/W̄ S̄T̄R̄B̄ READY · HOLD HOLDA BR SYNC

TMS320C25 — Memory Map

Program Space · Data Space · I/O Space · Harvard Dual-Bus

PROGRAM MEMORY 64K × 16-bit (0x0000–0xFFFF)

0x0000–0x0FFF	On-chip ROM (4K) (MP/$\overline{MC}$ = 0)
0x0000–0x0FFF	External Program (MP/$\overline{MC}$ = 1)
0x1000–0xEFFF	External Program Memory (60K words)
0xF000–0xFFFF	RAM B0 (CNFP) or External (CNFD) 256 words

B0 configured as program with CNFP.
MP/$\overline{MC}$=0 maps on-chip ROM to lower 4K.

DATA MEMORY 64K × 16-bit (0x0000–0xFFFF)

0x0000–0x005F	Memory-Mapped Registers DRR DXR TIM PRD IMR GREG
0x0060–0x007F	RAM Block B2 32 × 16-bit (data only)
0x0080–0x00FF	Reserved
0x0100–0x02FF	RAM Block B0 256 × 16-bit (when CNFD)
0x0300–0x03FF	RAM Block B1 256 × 16-bit (data only)
0x0400–0x07FF	Reserved
0x0800–0xFFFF	External Data Memory 63.5K × 16-bit

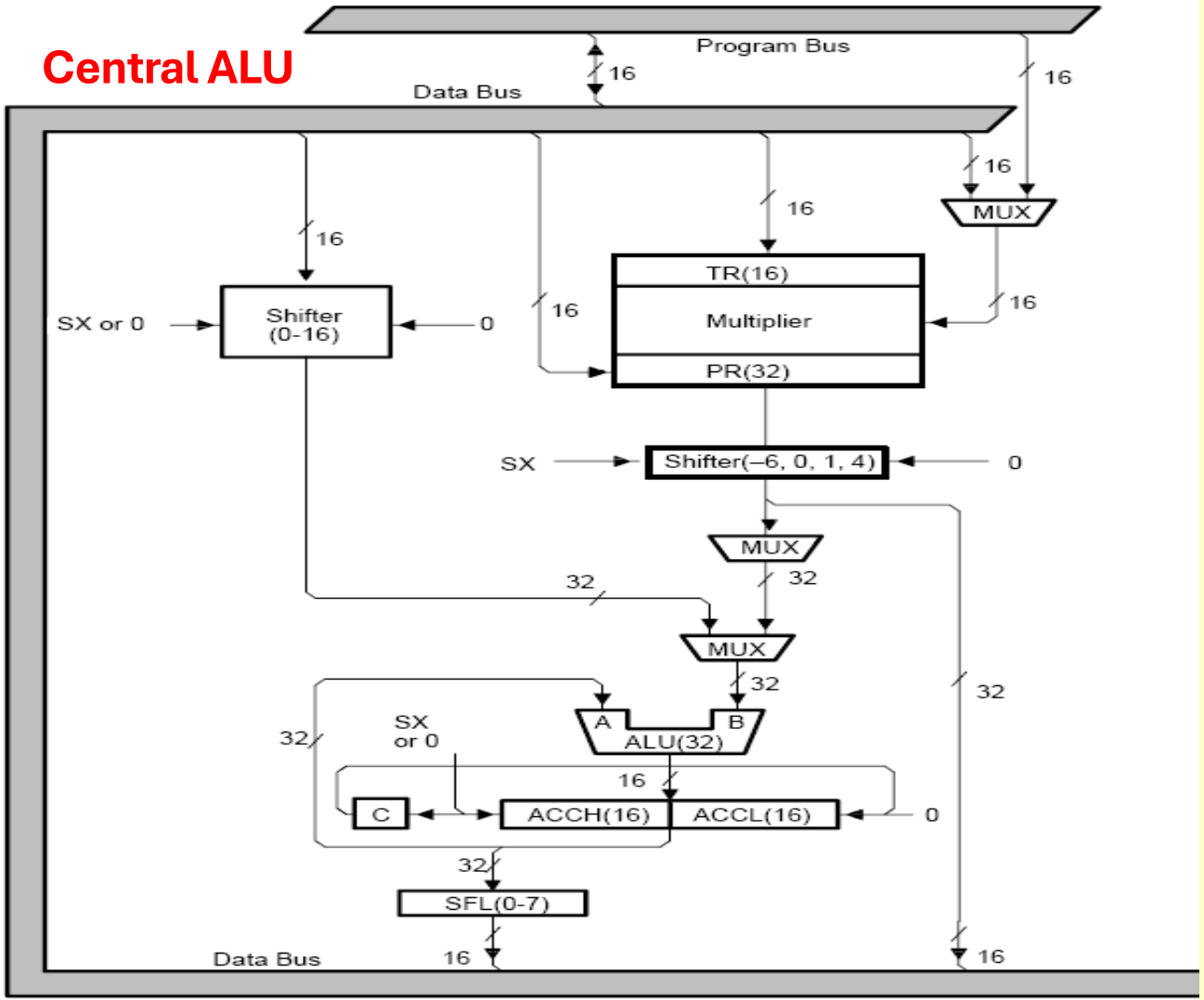
I/O SPACE 64K × 16-bit (0x0000–0xFFFF)

IN port#	Input Ports (0–0xFFFF) IN Daddr, port# Reads external device
OUT port#	Output Ports (0–0xFFFF) OUT Daddr, port# Writes external device

Key Architecture Facts

- 3 spaces: Program, Data, I/O — selected by $\overline{PS}$, $\overline{DS}$, $\overline{IS}$ pins
- Harvard: program fetch + data access simultaneously
- Direct addr: DP(9)+7-bit offset → 16-bit data address
- Indirect: 8 Aux Regs via ARAU (+AR0, −AR0, ±1, bit-reversed)
- Block moves: BLKD BLKP TBLR / TBLW instructions
- GREG: global memory allocation for multiprocessor designs

Central ALU


Program Bus
16
Data Bus
16
16
MUX
16
16
TR(16)
Multiplier
PR(32)
16
SX or 0
Shifter
(0-16)
0
SX
Shifter(–6, 0, 1, 4)
0
MUX
32
32
MUX
32
32
32
SX
or 0
A
B
ALU(32)
16
C
ACCH(16)
ACCL(16)
0
32
SFL(0-7)
Data Bus
16
16

TMS320C2x — Key Registers Overview

CPU / Arithmetic Registers		
ACC	32-bit	Accumulator — stores main computation result
T	16-bit	Temp register, first operand for multiplier
P	32-bit	Product register: result of T × operand
PC	16-bit	Program Counter
SP	16-bit	Stack Pointer (8-level hardware

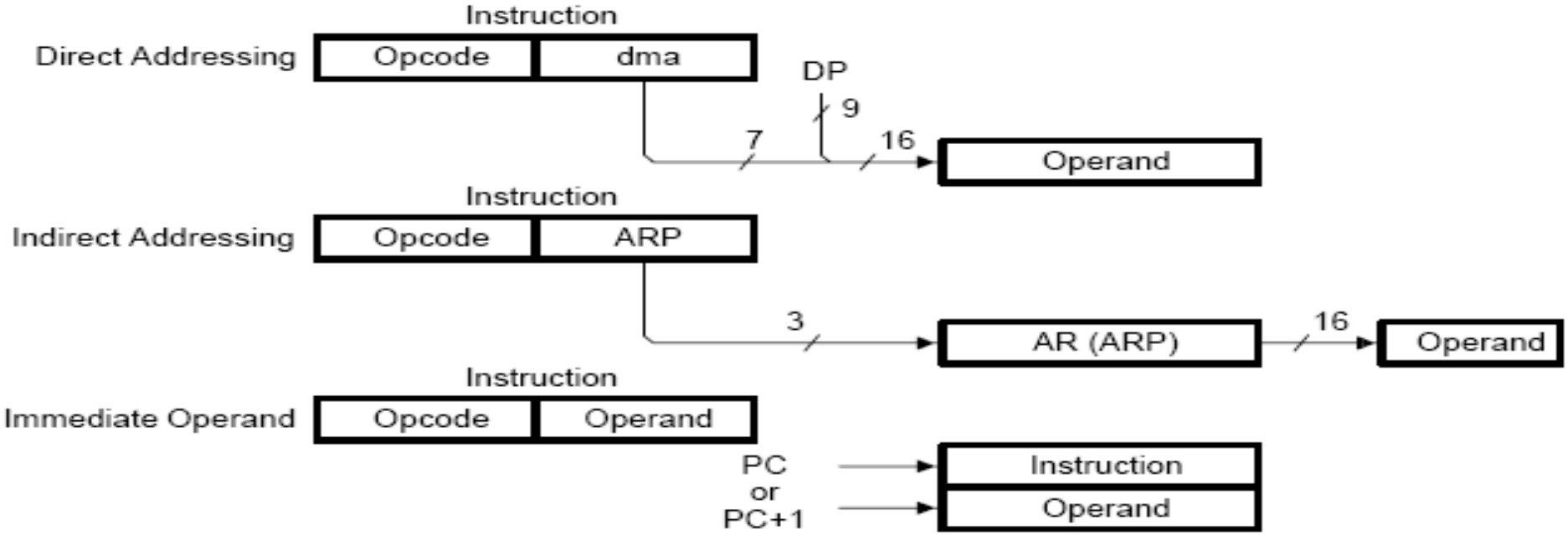
Address & Control Registers		
AR0-AR7	8×16-bit	Auxiliary Registers for indirect addressing
ARP	3-bit	Auxiliary Register Pointer (selects AR0-7)
DP	9-bit	Data Page Pointer: address bits[15:7]; =0 on RESET
ST0	16-bit	Status 0: OV, OVM, DP[8:0], ARP[2:0]

On-chip Data Memory Blocks		
B0 (256 w)	0x0200-0x02FF	Configurable: data OR program memory (CNFP)
B1 (256 w)	0x0300-0x03FF	Always data memory
B2 (32 w)	0x0060-0x007F	Always data memory (AR file mapped here)

Memory Addressing Modes

The TMS320C2x instruction set provides three memory addressing modes:

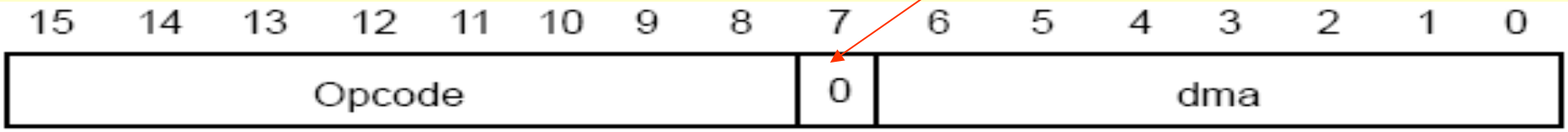
1. Direct addressing mode
2. Indirect addressing mode
3. Immediate addressing mode

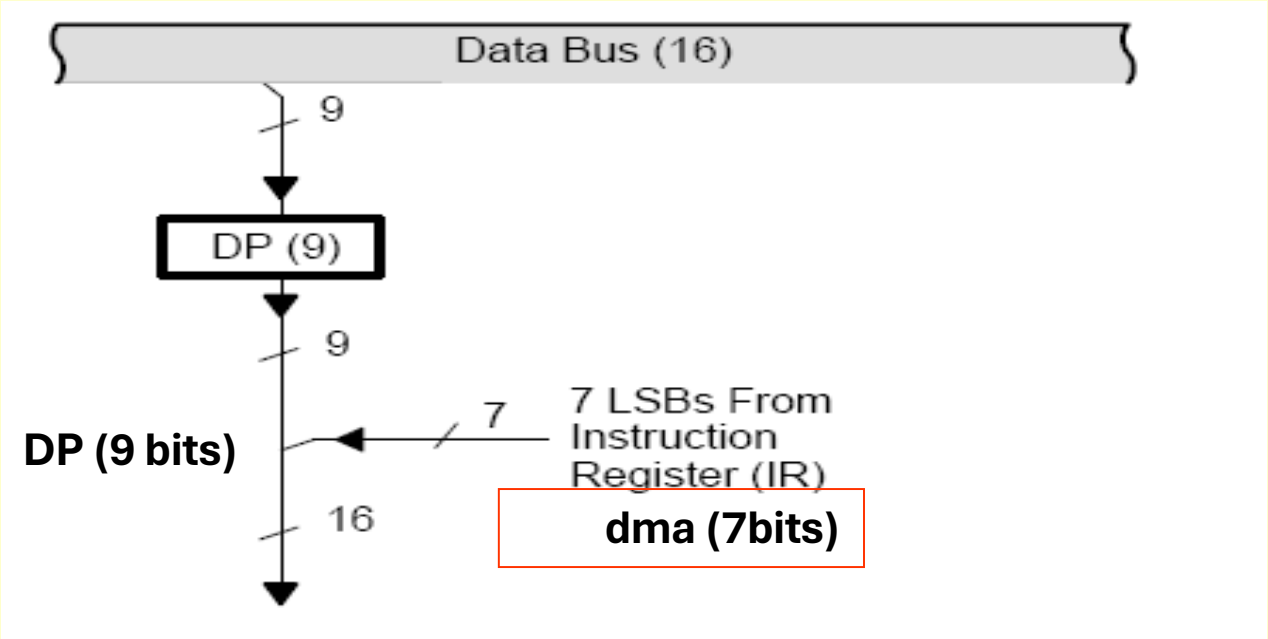


1.Direct Addressing Mode

- specific for all instruction excepting the CALL, branch, immediate instr. and no operand instruction

- **Instructions format (coding) using direct addressing mode:**





The processor loads the DP register through the:

LDP (load data memory page pointer),
LDPK (load data memory page-pointer immediate),
LST (load status register ST0) instructions.

DP =0 after RESET.

Ex1. Write the instructions that make: $(AC_{32})=(AC_{32})+(0695h)*8$, using direct addressing mode.

0695h = 0000 01101 001 0101

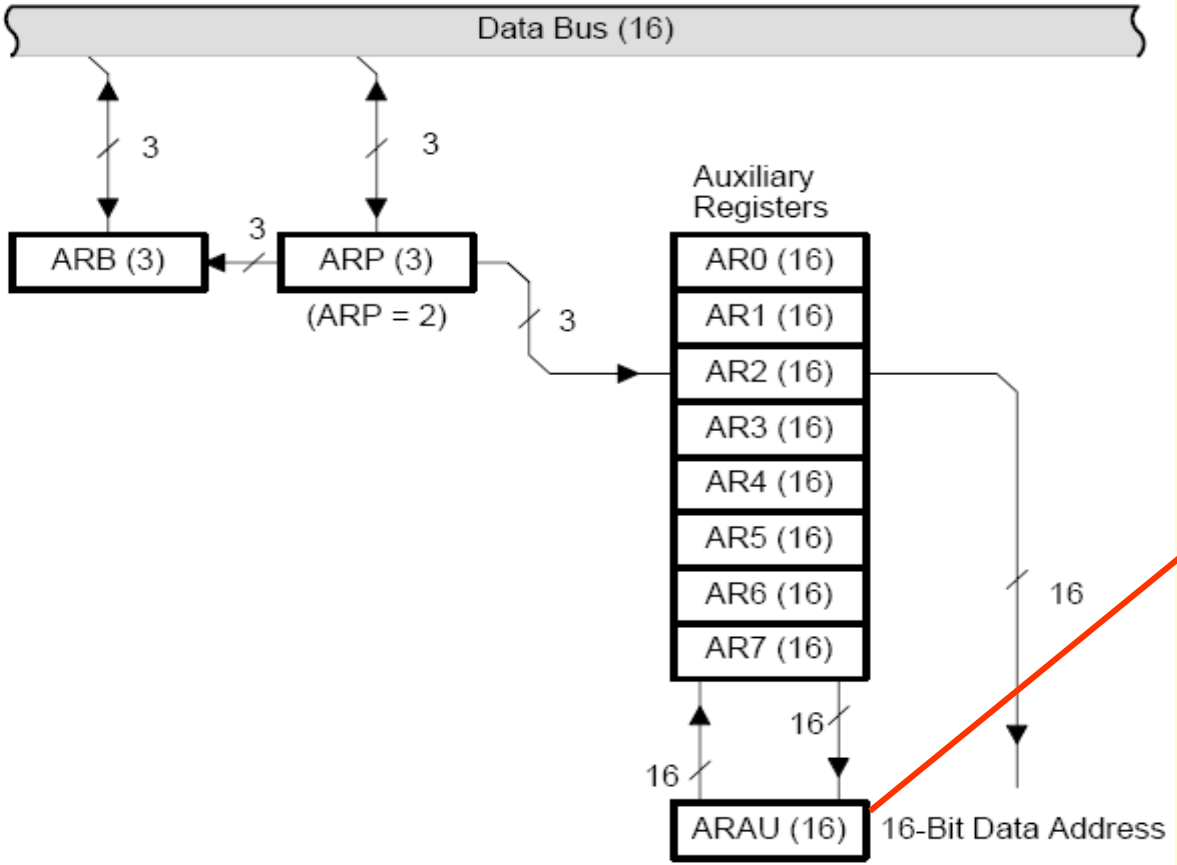
DP=13 (0Dh) dma= 21=15h

LDPK 13 ; DP=13=0dh

ADD 21,3 ; $(AC_{32})=(AC_{32})+(21)*8$

2. Indirect Addressing Mode

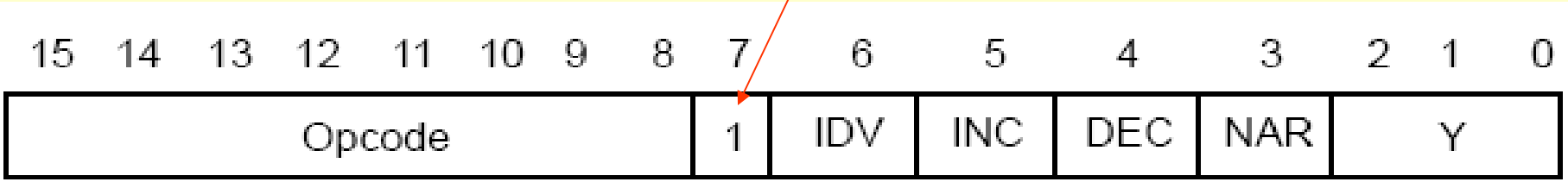
Indirect Addressing Block Diagram



- *
- *–
- *+
- *0–
- *0+
- *BR0–
- *BR0+

Indirect Addressing Mode

- The instructions format using indirect addressing mode



Bits 6	5	4	Arithmetic Operation
0	0	0	No operation on AR(ARP)
0	0	1	AR(ARP) – 1 → AR(ARP)
0	1	0	AR(ARP) + 1 → AR(ARP)
0	1	1	Reserved
1	0	0	AR(ARP) – AR0 → AR(ARP) [reverse carry propagation]
1	0	1	AR(ARP) – AR0 → AR(ARP)
1	1	0	AR(ARP) + AR0 → AR(ARP)
1	1	1	AR(ARP) + AR0 → AR(ARP) [reverse carry propagation]

ARs can be loaded by the instructions:

LAR (load auxiliary register),
LARK (load auxiliary register immediate),
LRLK (load auxiliary register long immediate).

and **can be modified** by :

ADRK (add to auxiliary register short immediate),
SBRK (subtract from auxiliary register short immediate)
MAR (modify auxiliary register) instruction or similar by the indirect addressing field of any instruction supporting indirect addressing.

Ex2. Write the instructions that make: $(AC_{32})=(AC_{32})+(0695h)*8$ using indirect addressing mode (AR3, ARP=011).

```
LARP 3          ;ARP=3

LRLK AR3,695h   ;AR3=0695h

ADD *,3         ;(AC)=(AC)+(695h)*8
```

Bits			Arithmetic Operation	Symbol/syntax
6	**5**	**4**		
0	0	0	No operation on AR(ARP)	*
0	0	1	AR(ARP) – 1 → AR(ARP)	*-
0	1	0	AR(ARP) + 1 → AR(ARP)	*+
0	1	1	Reserved	-
1	0	0	AR(ARP) – AR0 → AR(ARP) [reverse carry propagation]	*BR0-
1	0	1	AR(ARP) – AR0 → AR(ARP)	*0-
1	1	0	AR(ARP) + AR0 → AR(ARP)	*0+
1	1	1	AR(ARP) + AR0 → AR(ARP) [reverse carry propagation]	*BR0+

Ex3.

ADD *+,8 Add to the accumulator the contents of the data memory address defined by the contents of the current auxiliary register (AR). This data is left-shifted 8 bits before being added (*256). The current auxiliary register is auto-incremented by one
(ACC)=(ACC)+(AR)*2^8 and AR=AR+1

Ex4.

ADD *,8 As in Ex. 3, but with no auto-increment;

ADD *–,8 As in Ex. 3, except that the current auxiliary register is auto-decrement by 1 (AR=AR-1)

ADD *0+,8 works like in Ex. 3, except that the current auxiliary register receives the content of auxiliary register AR0 added to it (AR=AR+AR0).

ADD *0–,8 operates like Example 3, with the current auxiliary register subtracting the content of auxiliary register AR0.

ADD *+,8,3 As in Example 3, except that the auxiliary register pointer (ARP) is loaded with the new value 3 for subsequent instructions (ARP=3);

ADD *BR0–,8 Subtract the contents of auxiliary register AR0 from the current auxiliary register with reverse carry propagation.

ADD *BR0+,8 The content of auxiliary register AR0 is added to the current auxiliary register with reverse carry propagation.

How the FFT works (review)

- Basically, the computational problem for the DFT is to compute the sequence {*X*(*k*)} of *N* complex numbers, given another sequence of data {*x*(*n*)} of length *N*, according to the formula:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad 0 \le k \le N-1$$

$$e^{ix} = \cos(x) + i\sin(x)$$

$$W_N = e^{-j2\pi/N}$$

$$\text{Symmetry property}: W_N^{k+N/2} = -W_N^k$$

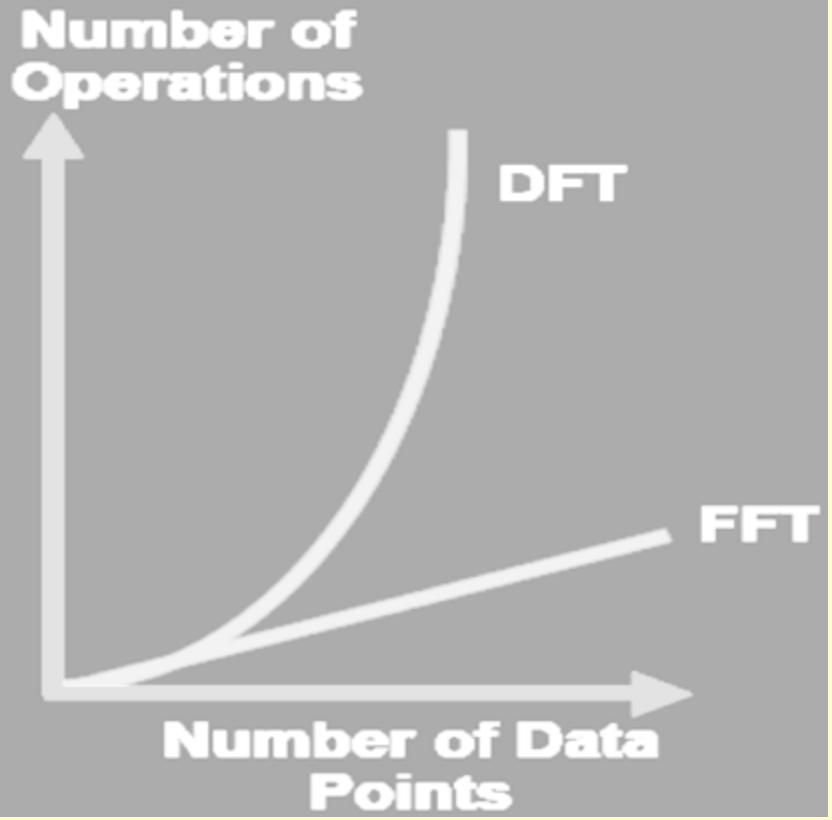
$$\text{Periodicity property}: W_N^{k+N} = W_N^k$$

In general, we assume the data sequence x(n) to be complex valued.
Similarly, the IDFT becomes:

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} X(k) W_N^{-nk}, \qquad 0 \le n \le N-1$$

Fast Fourier Transform

N	DFT	FFT
16	256	64
32	1,024	160
64	4,096	384
128	16,384	896
256	65,536	2,048
512	262,144	4,608
1,024	1,048,576	10,240
2,048	4,194,304	22,528
4,096	16,777,216	49,152



DFT vs FFT — Computational Complexity

DFT direct computation:

N^2 complex multiplications
$N(N-1)$ complex additions

Cooley-Tukey Radix-2 FFT:

$(N/2)\cdot\log_2 N$ complex multiplications
$N\cdot\log_2 N$ complex additions

Speedup ≈ **$2N / \log_2 N$** (e.g. N=1024 → speedup ≈ 204×)

Complex Multiplications			
N	DFT (N^2)	FFT ($N/2\cdot\log_2 N$)	Speed up
8	64	12	5×
64	4,096	192	21×
256	65,536	1,024	64×
1024	1,048,576	5,120	204×
4096	16,777,216	24,576	682×

- *The FFT is therefore the preferred algorithm for real-time DSP applications on the TMS320C2x, where both computation speed and memory efficiency are critical constraints.*

FFT Foundation: Real DFT vs. Complex DFT

The FFT operates on the Complex DFT

Real DFT

Takes an N-point time-domain signal and produces N/2 +1 frequency-domain output points.

Complex DFT

Takes two N-point signals (real + imaginary) and produces N output frequency points.

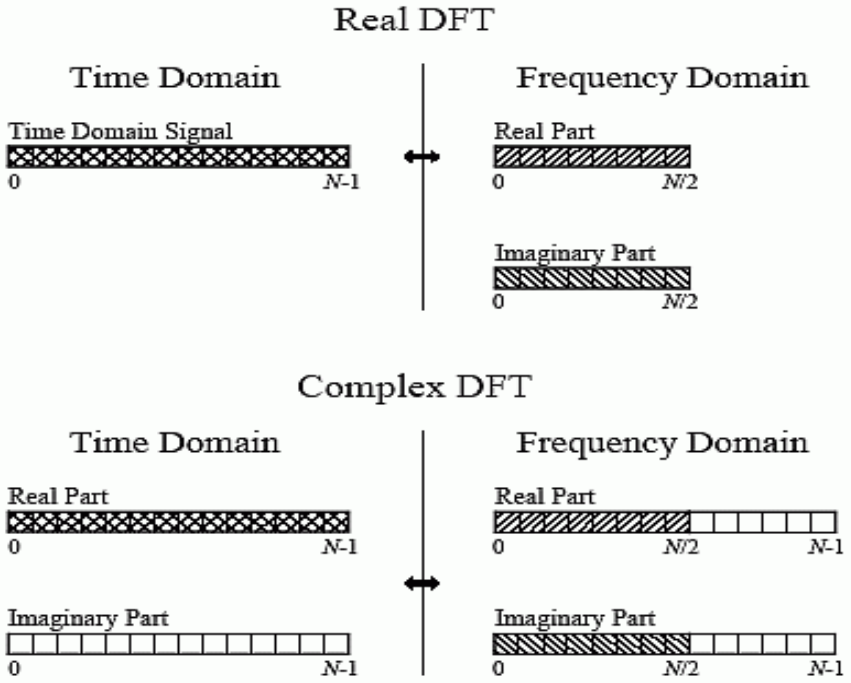


FIGURE 12-1
Comparing the real and complex DFTs. The real DFT takes an N point time domain signal and creates two $N/2+1$ point frequency domain signals. The complex DFT takes two N point time domain signals and creates two N point frequency domain signals. The crosshatched regions shows the values common to the two transforms.

FFT Algorithm — Three Steps

1

Time Domain Decomposition

Decompose the N-point time-domain signal into N single-point signals using interlace decomposition (bit-reversal).

2

Compute Frequency Spectra

Calculate the N frequency spectra corresponding to the N single-point time-domain signals. No computation needed — each 1-point signal is already its own spectrum.

3

Synthesize the Spectrum

Combine the N single-point frequency spectra stage by stage (reverse of decomposition) into a single N-point frequency spectrum.

STEP 1a Time Domain Decomposition

Interlace Decomposition

- An N-point signal is split into N single-point signals through repeated interlace decomposition.
- Each stage separates even-indexed samples from odd-indexed samples.
- The number of stages required is $\log_2(N)$:

Stage Requirements

N = 16 (2^4) → 4 stages

N = 512 (2^9) → 9 stages

N = 4096 (2^{12}) → 12 stages

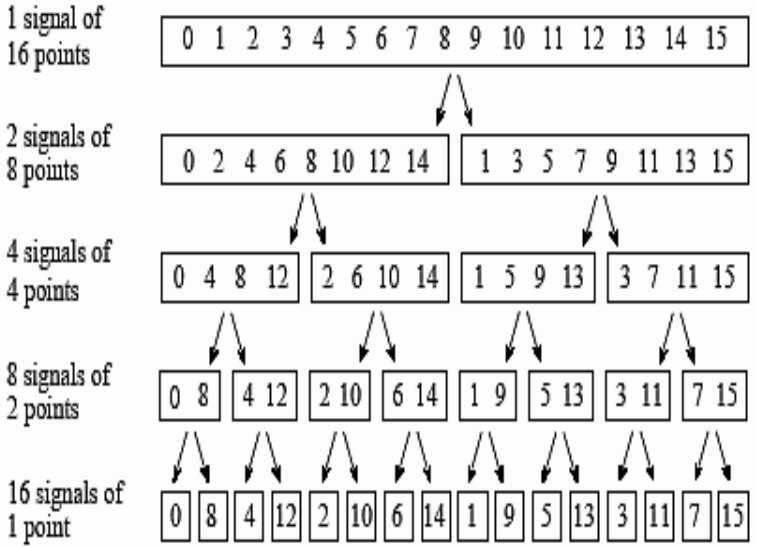


FIGURE 12-2
The FFT decomposition. An N point signal is decomposed into N signals each containing a single point. Each stage uses an *interlace decomposition*, separating the even and odd numbered samples.

STEP 1b Bit-Reversal Sorting

How Bit-Reversal Works

The decomposition reorders samples by reversing the binary representation of each sample's index.

Example (4-bit, N=16):

0000 → 0 (stays at 0)

1000 → 8 (moves to 1)

0100 → 4 (moves to 2)

1100 → 12 (moves to 3)

- A bit-reversal sorting algorithm implements this reordering efficiently — no recursive calls needed.

Sample numbers in normal order		Sample numbers after bit reversal	
Decimal	*Binary*	*Decimal*	*Binary*
0	0000	0	0000
1	0001	8	1000
2	0010	4	0100
3	0011	12	1100
4	0100	2	0010
5	0101	10	1010
6	0110	6	0100
7	0111	14	1110
8	1000	1	0001
9	1001	9	1001
10	1010	5	0101
11	1011	13	1101
12	1100	3	0011
13	1101	11	1011
14	1110	7	0111
15	1111	15	1111

FIGURE 12-3
The FFT bit reversal sorting. The FFT time domain decomposition can be implemented by sorting the samples according to bit reversed order.

STEP 2 Compute N Frequency Spectra

No computation required!

① **1-Point Signals**

After decomposition, we have N signals — each containing exactly one sample point.

② **Frequency = Value**

The frequency spectrum of a 1-point signal is the signal itself. The DFT of a single value is just that value.

③ **Identity Transform**

Each single-point time-domain signal is simultaneously its own 1-point frequency-domain spectrum.

STEP 3 Synthesize the Frequency Spectrum

Combining spectra stage by stage (16-point example):

Stage 1	16 spectra × 1 pt → 8 spectra × 2 pts
Stage 2	8 spectra × 2 pts → 4 spectra × 4 pts
Stage 3	4 spectra × 4 pts → 2 spectra × 8 pts
Stage 4	2 spectra × 8 pts → 1 spectrum × 16 pts

⚠ Bit-reversal does NOT work for synthesis — stages must be processed in strict sequential order.

STEP 3 Synthesis: Duplication and Sinusoidal Shift

How spectra are combined:

- Each spectrum is duplicated, then the duplicate is added to the original.
- If the corresponding time-domain signal was shifted by one sample, the odd spectrum is multiplied by a sinusoid (twiddle factor) before addition.
- This sinusoidal multiplication in the frequency domain corresponds to the time-domain shift.

Key insight: Diluting a time-domain signal with zeros causes its frequency spectrum to duplicate.

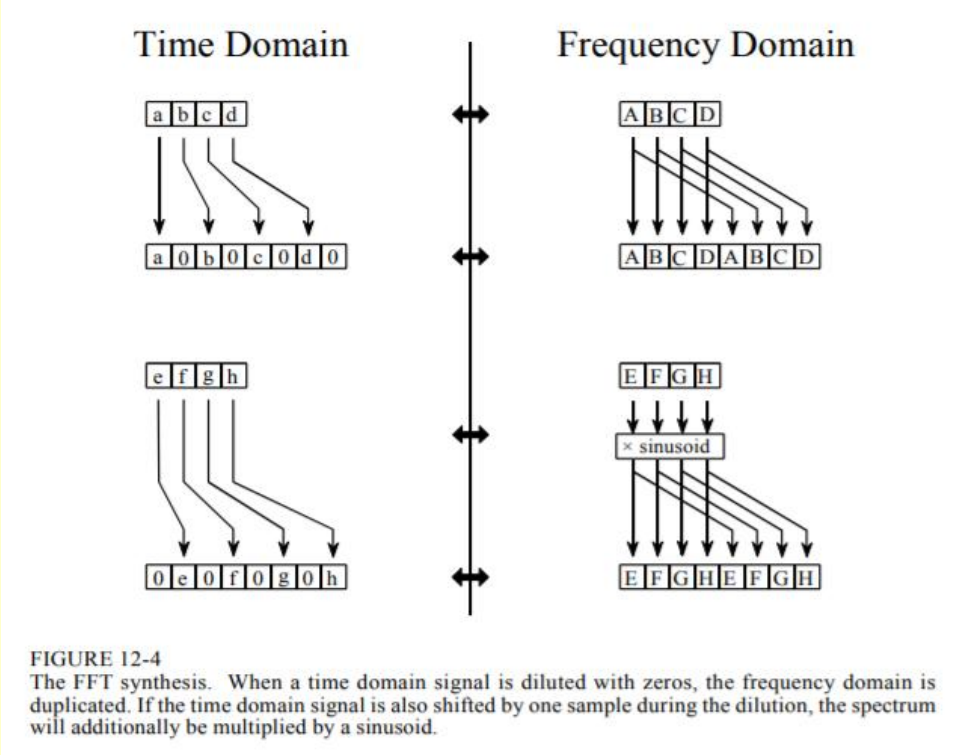
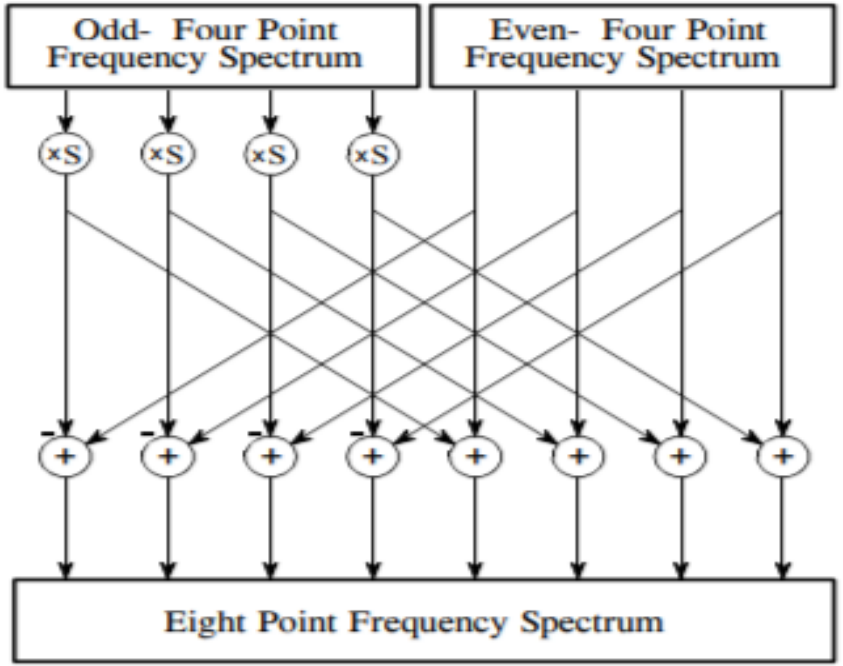


FIGURE 12-4
The FFT synthesis. When a time domain signal is diluted with zeros, the frequency domain is duplicated. If the time domain signal is also shifted by one sample during the dilution, the spectrum will additionally be multiplied by a sinusoid.

STEP 3 Synthesis Flow: Combining 4-Point Spectra into 8-Point

FIGURE 12-5
FFT synthesis flow diagram. This shows the method of combining two 4 point frequency spectra into a single 8 point frequency spectrum. The ×S operation means that the signal is multiplied by a sinusoid with an appropriately selected frequency.



- The ×S operation: a sinusoid multiplied each odd-spectrum point at a selected frequency before being added to the even-spectrum counterpart.

The FFT Butterfly

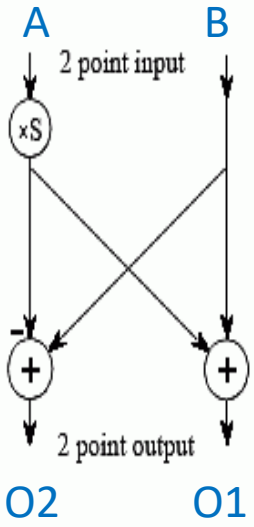


FIGURE 12-6
The FFT butterfly. This is the basic calculation element in the FFT, taking two complex points and converting them into two other complex points.

Basic Computational Element

The butterfly takes 2 complex inputs (A, B) and a fixed coefficient (W) and produces 2 complex outputs:

- Complex variables A and B as inputs/outputs
- Fixed twiddle-factor coefficients W = e^(−j2πk/N)
- Address calculations are complex (bit-reversed order)
- Requires fast memory access for registers, address pointers, and constants
- Core operations: one complex multiply + two complex additions

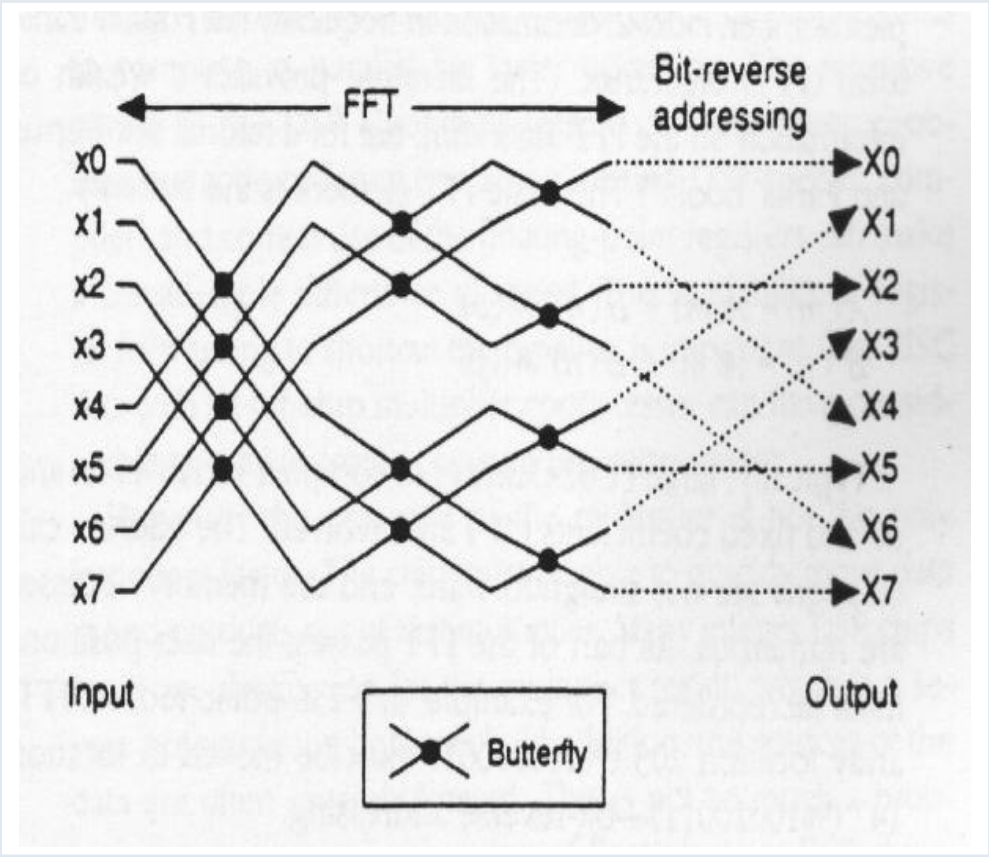
$\text{Output}_1 = A + W \cdot B \quad \text{Output}_2 = A - W \cdot B$

DSP Addressing: Bit-Reversal in Hardware

FFTs produce output in bit-reversed order — a hardware addressing challenge:

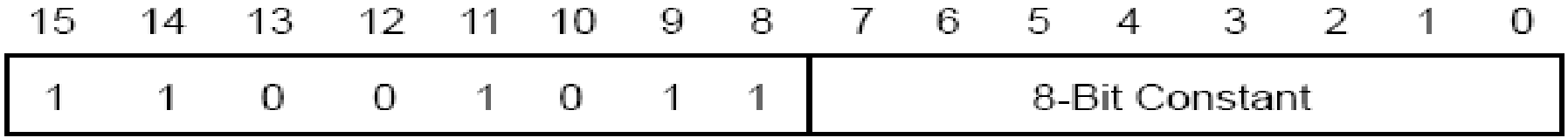
Normal Index	Binary	Bit-Reversed	Binary
0	000	0	000
1	001	4	100
2	010	2	010
3	011	6	110
4	100	1	001
5	101	5	101
6	110	3	011
7	111	7	111

Solution: Many DSPs provide a dedicated bit-reverse addressing mode for radix-2 FFT auto-incremented with no overhead.

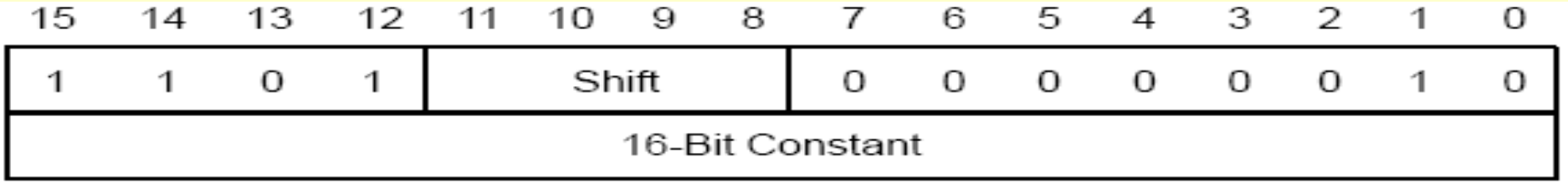


Immediate Addressing Mode

- In immediate addressing, the instruction word(s) contain the value of the immediate operand.
- The TMS320C2x has both single-word (8-bit /13-bit constant) *short immediate instructions*



and two-word (16-bit constant) *long immediate instructions*.



ADDK Add to accumulator short immediate (8-bit absolute constant)

ADRK Add to auxiliary register short immediate (8-bit absolute constant)

LACK Load accumulator short immediate (8-bit absolute constant)

LARK Load auxiliary register short immediate (8-bit absolute constant)

LARP Load auxiliary register pointer (3-bit constant)

LDPK Load data memory page pointer immediate (9-bit constant)

MPYK Multiply immediate (13-bit 2s-complement constant)

RPTK Repeat instruction as specified by immediate value (8-bit constant)

SBRK Subtract from auxiliary register short immediate (8-bit absolute constant)

SUBK Subtract from accumulator short immediate (8-bit absolute constant).

Ex.

RPTK 99 ; Execute the next-instruction following this instruction 100 x.
Next instr.;

ADLK Add to accumulator long immediate with shift (abs. or 2s complement)

ANDK AND immediate with accumulator with shift

LALK Load accumulator long immediate with shift (absolute or 2s complement)

LRLK Load auxiliary register long immediate

ORK OR immediate with accumulator with shift

SBLK Subtract from accumulator long immediate with shift (absolute or 2s complement)

XORK Exclusive-OR immediate with accumulator with shift.

EX.
ADLK 16384,2 ; (AC)=(AC)+16834*4

Warning!
Immediate addressing index for C5x/C54x is # !!! So:
C2X syntax: LDPK val
C5X: LDP # val ; the instr. Have the same effect

Instruction Set

Syntax

Direct: [label] **MNEMONIC** dma [, shift] ; [comments]

Indirect: [label] **MNEMONIC** {ind} [, shift [next ARP]]; {com]

{ * | * + | * – | * 0 + | * 0 – | * BRO + | * BRO –}

Immediate: [label] **MNEMONIC** [constant] ; [com]

Operands

$0 \leq$ dma ≤ 127
$0 \leq$ next ARP ≤ 7
$0 \leq$ constant ≤ 255

Execution :

(PC) + 1 → PC
(ACC) + [(dma) × 2^{shift}] →ACC
If SXM = 1: Then (dma) is sign-extended.
If SXM = 0: Then (dma) is not sign-extended.
Affects OV; affected by OVM and SXM.
Affects C.

INSTRUCTIONS CLASSIFICATION

- Accumulator memory reference instructions
- Auxiliary registers and data-page pointer instructions
- T register, P register, and multiply instructions
- Branch/call instructions
- I/O and data memory operations
- Control instructions

- The instruction set for the TMS320C2x processor is a superset of the TMS320C1x instruction set. Included in the instruction set are four special groups of instructions to improve overall processor throughput and ease of use:

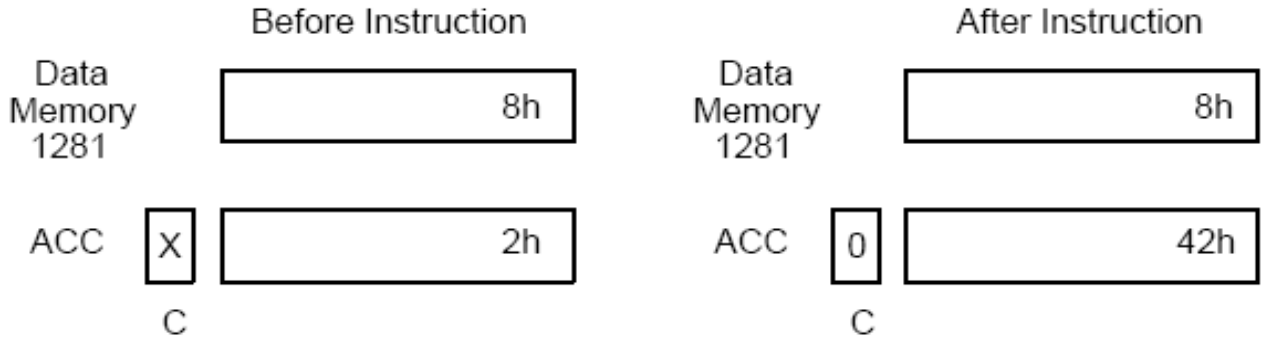
- Extended-precision arithmetic (ADDC, SUBB, MPYU, BC, BNC, SC, and RC)
- Adaptive filtering (MPYA, MPYS, and ZALR)
- Control and I/O (RHM, SHM, RTC, STC, RFSM, and SFSM)
- Accumulator and registers (SPH, SPL, ADDK, SUBK, ADRK, SBRK, ROL and ROR).

Example

```
ADD   DAT1,3   ;(DP = 10)
```

or

```
ADD   *,3      ;If current auxiliary register contains 1281.
```



ACCUMULATOR MEMORY REFERENCE INSTRUCTIONS						
Mnemonic and Description		Words	16-Bit Opcode MSB			LSB
ABS	Absolute value of accumulator	1	1100	1110	0001	1011
ADD	Add to accumulator with shift	1	0000	SSSS	MDDD	DDDD
ADDC	Add to accumulator with carry	1	0100	0011	MDDD	DDDD
ADDH	Add to high accumulator	1	0100	1000	MDDD	DDDD
ADDK	Add to accumulator short immediate	1	1100	1100	KKKK	KKKK
ADDS	Add to low accumulator with sign-extension suppressed	1	0100	1001	MDDD	DDDD
ADDT	Add to accumulator with shift specified by T register	1	0100	1010	MDDD	DDDD
ADLK	Add to accumulator long immediate with shift	2	1101	SSSS	0000	0010
AND	AND with accumulator	1	0100	1110	MDDD	DDDD
ANDK	AND immediate with accumulator with shift	2	1101	SSSS	0000	0100
CMPL	Complement accumulator	1	1100	1110	0010	0111
LAC	Load accumulator with shift	1	0010	SSSS	MDDD	DDDD
LACK	Load accumulator short immediate	1	1100	1010	KKKK	KKKK
LACT	Load accumulator with shift specified by T register	1	0100	0010	MDDD	DDDD
LALK	Load accumulator long immediate with shift	2	1101	SSSS	0000	0001
NEG	Negate accumulator	1	1100	1110	0010	0011
NORM	Normalize contents of accumulator	1	1100	1110	1010	0010
OR	OR with accumulator	1	0100	1101	MDDD	DDDD
ORK	OR immediate with accumulator with shift	2	1101	SSSS	0000	0101
ROL	Rotate accumulator left	1	1100	1110	0011	0100
ROR	Rotate accumulator right	1	1100	1110	0011	0101

ACCUMULATOR MEMORY REFERENCE INSTRUCTIONS						
Mnemonic and Description		Words	16-Bit Opcode MSB			LSB
SAC	Store high accumulator with shift	1	0110	1XXX	MDDD	DDDD
SACL	Store low accumulator with shift	1	0110	0XXX	MDDD	DDDD
SBLK	Subtract from accumulator long immediate with shift	2	1101	SSSS	0000	0011
SFL	Shift accumulator left	1	1100	1110	0001	1000
SFR	Shift accumulator right	1	1100	1110	0001	1001
SUB	Subtract from accumulator with shift	1	0001	SSSS	MDDD	DDDD
SUBB	Subtract from accumulator with borrow	1	0100	1111	MDDD	DDDD
SUBC	Conditional subtract	1	0100	0111	MDDD	DDDD
SUBH	Subtract from high accumulator	1	0100	0100	MDDD	DDDD
SUBK	Subtract from accumulator short immediate	1	1100	1101	KKKK	KKKK
SUBS	Subtract from low accumulator with sign extension suppressed		0100	0101	MDDD	DDDD
SUBT	Subtract from accumulator with shift specified by T register	1	0100	0110	MDDD	DDDD
XOR	Exclusive-OR with accumulator	1	0100	1100	MDDD	DDDD
XORK	Exclusive-OR immediate with accumulator with shift	2	1101	SSSS	0000	0110
ZAC	Zero accumulator	1	1100	1010	0000	0000
ZALH	Zero low accumulator and load high accumulator	1	0100	0000	MDDD	DDDD
ZALR	Zero low accumulator and load high accumulator with rounding	1	0111	1011	MDDD	DDDD
ZALS	Zero accumulator and load low accumulator with sign extension suppressed	1	0100	0001	MDDD	DDDD

AUXILIARY REGISTERS AND DATA PAGE POINTER INSTRUCTIONS						
Mnemonic and Description		Words	16-Bit Opcode MSB			LSB
ADRK	Add to auxiliary register short immediate	1	0111	1110	KKKK	KKKK
CMPR	Compare auxiliary register with auxiliary register AR0	1	1100	1110	0101	00KK
LAR	Load auxiliary register	1	0011	0RRR	MDDD	DDDD
LARK	Load auxiliary register short immediate	1	1100	0RRR	KKKK	KKKK
LARP	Load auxiliary register pointer	1	0101	0101	1000	1RRR
LDP	Load data memory page pointer	1	0101	0010	MDDD	DDDD
LDPK	Load data memory page pointer immediate	1	1100	100K	KKKK	KKKK
LRLK	Load auxiliary register long immediate	2	1101	0RRR	0000	0000
MAR	Modify auxiliary register	1	0101	0101	MDDD	DDDD
SAR	Store auxiliary register	1	0111	0RRR	MDDD	DDDD
SBRK	Subtract from auxiliary register short immediate	1	0111	1111	KKKK	KKKK

T REGISTER, P REGISTER, AND MULTIPLY INSTRUCTIONS

Mnemonic and Description		Words	16-Bit Opcode MSB			LSB
APAC	Add P register to accumulator	1	1100	1110	0001	0101
LPH	Load high P register	1	0101	0011	MDDD	DDDD
LT	Load T register	1	0011	1100	MDDD	DDDD
LTA	Load T register and accumulate previous product	1	0011	1101	MDDD	DDDD
LTD	Load T register, accumulate previous product and move data	1	0011	1111	MDDD	DDDD
LTP	Load T register and store P register in accumulator	1	0011	1110	MDDD	DDDD
LTS	Load T register and subtract previous product	1	0101	1011	MDDD	DDDD
MAC	Multiply and accumulate	2	0101	1101	MDDD	DDDD
MACD	Multiply and accumulate with data move	2	0101	1100	MDDD	DDDD
MPY	Multiply (with T register, store product in P register)	1	0011	1000	MDDD	DDDD
MPYA	Multiply and accumulate previous product	1	0011	1010	MDDD	DDDD
MPYK	Multiply immediate	1	101K	KKKK	KKKK	KKKK
MPYS	Multiply and subtract previous product	1	0011	1011	MDDD	DDDD
MPYU	Multiply unsigned	1	1100	1111	MDDD	DDDD
PAC	Load accumulator with P register	1	1100	1110	0001	0100
SPAC	Subtract P register from accumulator	1	1100	1110	0001	0110
SPH	Store high P register	1	0111	1101	MDDD	DDDD
SPL	Store low P register	1	0111	1100	MDDD	DDDD
SPM	Set P register output shift mode	1	1100	1110	0000	10KK
SQRA	Square and accumulate	1	0011	1001	MDDD	DDDD
SQRS	Square and subtract previous product	1	0101	1010	MDDD	DDDD

BRANCH/CALL INSTRUCTIONS

Mnemonic and Description		Words	16-Bit Opcode MSB			LSB
B	Branch unconditionally	2	1111	1111	1DDD	DDDD
BACC	Branch to address specified by accumulator	1	1100	1110	0010	0101
BANZ	Branch on auxiliary register not zero	2	1111	1011	1DDD	DDDD
BBNZ	Branch if TC bit ≠ 0	2	1111	1001	1DDD	DDDD
BBZ	Branch if TC bit = 0	2	1111	1000	1DDD	DDDD
BC	Branch on carry	2	0101	1110	1DDD	DDDD
BGEZ	Branch if accumulator ≠ 0	2	1111	0100	1DDD	DDDD
BGZ	Branch if accumulator > 0	2	1111	0001	1DDD	DDDD
BIOZ	Branch on I/O status = 0	2	1111	1010	1DDD	DDDD
BLEZ	Branch if accumulator ≤ 0	2	1111	0010	1DDD	DDDD
BLZ	Branch if accumulator < 0	2	1111	0011	1DDD	DDDD
BNC	Branch on no carry	2	0101	1111	1DDD	DDDD
BNV	Branch if no overflow	2	1111	0111	1DDD	DDDD
BNZ	Branch if accumulator ≠ 0	2	1111	0101	1DDD	DDDD
BV	Branch on overflow	2	1111	0000	1DDD	DDDD
BZ	Branch if accumulator = 0	2	1111	0110	1DDD	DDDD
CALA	Call subroutine indirect	1	1100	1110	0010	0100
CALL	Call subroutine	2	1111	1110	1DDD	DDDD
RET	Return from subroutine	1	1100	1110	0010	0110
TRAP	Software interrupt	1	1100	1110	0001	1110

I/O AND DATA MEMORY OPERATIONS

Mnemonic and Description		Words	16-Bit Opcode MSB			LSB
BLKD	Block move from data memory to data memory	2	1111	1101	MDDD	DDDD
BLKP	Block move from program memory to data memory	2	1111	1100	MDDD	DDDD
DMOV	Data move in data memory	1	0101	0110	MDDD	DDDD
FORT	Format serial port registers	1	1100	1110	0000	111K
IN	Input data from port	1	1000	AAAA	MDDD	DDDD
OUT	Output data to port	1	1110	AAAA	MDDD	DDDD
RFSM	Reset serial port frame synchronization mode	1	1100	1110	0011	0110
RTXM	Reset serial port transmit mode	1	1100	1110	0010	0000
RXF	Reset external flag	1	1100	1110	0000	1100
SFSM	Set serial port frame synchronization mode	1	1100	1110	0011	0111
STXM	Set serial port transmit mode	1	1100	1110	0010	0001
SXF	Set external flag	1	1100	1110	0000	1101
TBLR	Table read	1	0100	1000	MDDD	DDDD
TBLW	Table write	1	0101	1001	MDDD	DDDD

CONTROL INSTRUCTIONS						
Mnemonic and Description		Words	16-Bit Opcode MSB			LSB
BIT	Test bit	1	1001	BBBB	MDDD	DDDD
BITT	Test bit specified by T register	1	0101	0111	MDDD	DDDD
CNFD†	Configure block as data memory	1	1100	1110	0000	0100
CNFP†	Configure block as program memory	1	1100	1110	0000	0101
CONF†	Configure block as data/program memory	1	1100	1110	0011	11KK
DINT	Disable interrupt	1	1100	1110	0000	0001
EINT	Enable interrupt	1	1100	1110	0000	0000
IDLE	Idle until interrupt	1	1100	1110	0001	1111
LST	Load status register ST0	1	0101	0000	MDDD	DDDD
LST1	Load status register ST1	1	0101	0001	MDDD	DDDD
NOP	No operation	1	0101	0101	0000	0000
POP	Pop top of stack to low accumulator	1	1100	1110	0001	1101
POPD	Pop top of stack to data memory	1	0111	1010	MDDD	DDDD
PSHD	Push data memory value onto stack	1	0101	0100	MDDD	DDDD
PUSH	Push low accumulator onto stack	1	1100	1110	0001	1100
RC	Reset carry bit	1	1100	1110	0011	0000
RHM	Reset hold mode	1	1100	1110	0011	1000
ROVM	Reset overflow mode	1	1100	1110	0000	0010
RPT	Repeat instruction as specified by data memory value	1	0100	1011	MDDD	DDDD
RPTK	Repeat instruction as specified by immediate value	1	1100	1011	KKKK	KKKK
RSXM	Reset sign-extension mode	1	1100	1110	0000	0110
RTC	Reset test/control flag	1	1100	1110	0011	0010
SC	Set carry bit	1	1100	1110	0011	0001
SHM	Set hold mode	1	1100	1110	0011	1001
SOVM	Set overflow mode	1	1100	1110	0000	0011
SST	Store status register ST0	1	0111	1000	MDDD	DDDD
SST1	Store status register ST1	1	0111	1001	MDDD	DDDD
SSXM	Set sign-extension mode	1	1100	1110	0000	0111
STC	Set test/control flag	1	1100	1110	0011	0011

TMS320 Family Comparison — C2x vs C5x / C54x

Syntax and Architecture Differences

Feature	TMS320C2x	TMS320C5x / C54x
Immediate addressing prefix	none (LDPK 5)	# required (LDP #5)
Accumulator width	32-bit (ACC)	32-bit (A), 40-bit (A+guard)
Multiply result	32-bit P register	32-bit (T×operand → A)
Auxiliary registers	AR0–AR7 (8 × 16-bit)	AR0–AR7 (8 × 16-bit)
Max clock rate (typical)	40 MIPS (C25)	100 MIPS (C5402)
On-chip RAM	544 words (B0+B1+B2)	Up to 16Kwords (C54x)
Program memory page	B0 via CNFP	Larger unified program space
Repeat instruction	RPTK n (8-bit)	RPT n (16-bit count)
Hardware stack depth	8 levels	Up to 256 levels (C54x)

Addressing Modes Summary

1. Direct Addressing

- 16-bit address = DP[15:7] || dma[6:0]
- DP (9 bits) selects one of 512 data pages
- dma (7 bits) → offset within the page (0–127)
- Load DP via: LDPK val or LDP #val (C5x)
- DP = 0 on RESET
- Syntax: INSTR dma [, shift]
- Example: ADD 21,3 ; ACC += mem[DP:21] << 3

```
LDPK  13
ADD   0x15, 3  ;
```

2. Indirect Addressing

- Uses ARP to select current AR (AR0–AR7)
- AR (16-bit) holds full effective address
- Auto-modify: * *+ *– *0+ *0– *BR0+ *BR0–
- BR0± uses bit-reversed arithmetic (for FFT)
- Supports next-ARP field in instruction word
- Syntax: INSTR {ind} [, shift [, nextARP]]
- Example: ADD *+, 8, 3 ; shift 8, ARP→3

```
LRLK  AR3, 0x695h
ADD   *, 3       ; ACC += mem[AR3] << 3
```

3. Immediate Addressing

- Value embedded in the instruction word
- Short (1 word): 8-bit or 13-bit constant
- Long (2 word): 16-bit constant (ADLK, LALK…)
- No memory read required → fastest mode
- Short immediates: ADDK, LACK, LDPK, RPTK…
- Long immediates: ADLK, LALK, LRLK, SBLK…
- Example: RPTK 99 ; repeat next instr. 100×

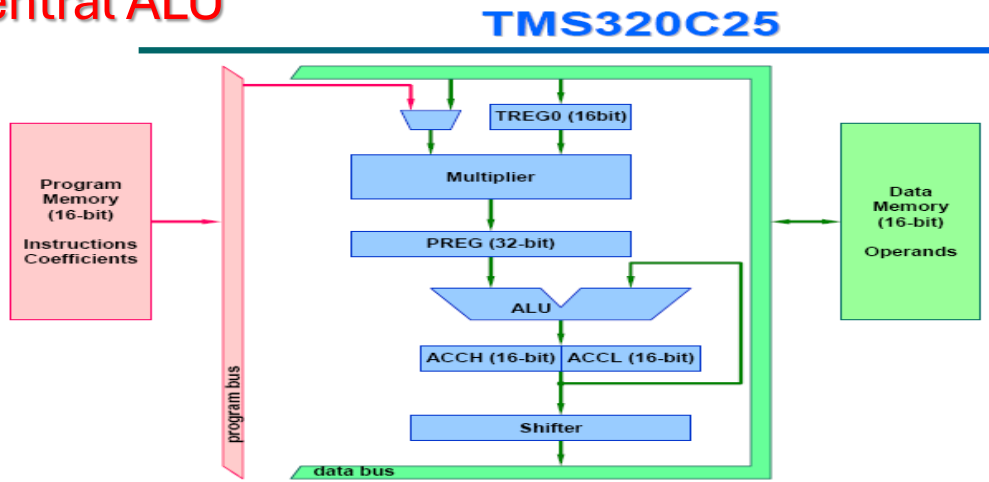
```
RPTK  99
MACD  h0, *-   ; 100-tap FIR loop body
```

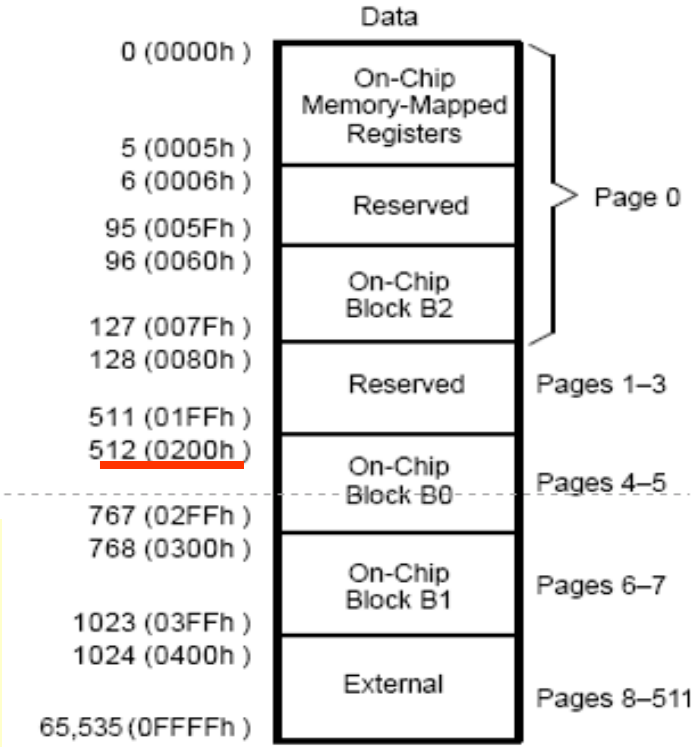
Syntax warning: C2x uses no prefix for immediates (e.g. LDPK 5), while C5x/C54x requires # (e.g. LDP #5). Both achieve the same effect but are NOT interchangeable.

Ex1. Write the program sequence in C2x A.L. which compute:

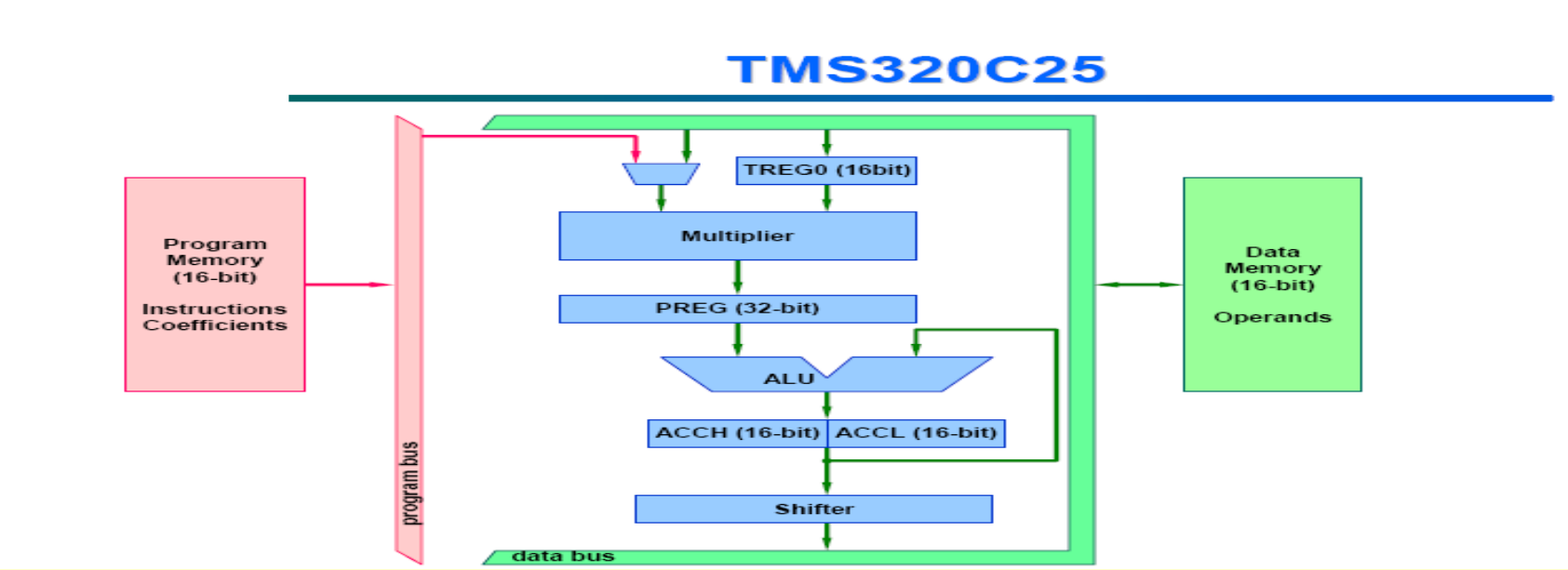
(202H, 203H) = (200H) * (201H) ; B0 data memory block

Central ALU





```
LARP 1                  ; ARP=1
LRLK  AR1,200h          ; AR1=200h, address data block B0
LT  *+                  ; T=(200h), AR1=201h
MPY *+                  ; P=T*(201h) , AR1=202h
PAC                     ;ACC=P, if PM=00   SPM 0
SACH*+                  ;(202h)=ACCH, AR1=203h
SACL*                   ;(203h)=ACCL ; big-endian
```

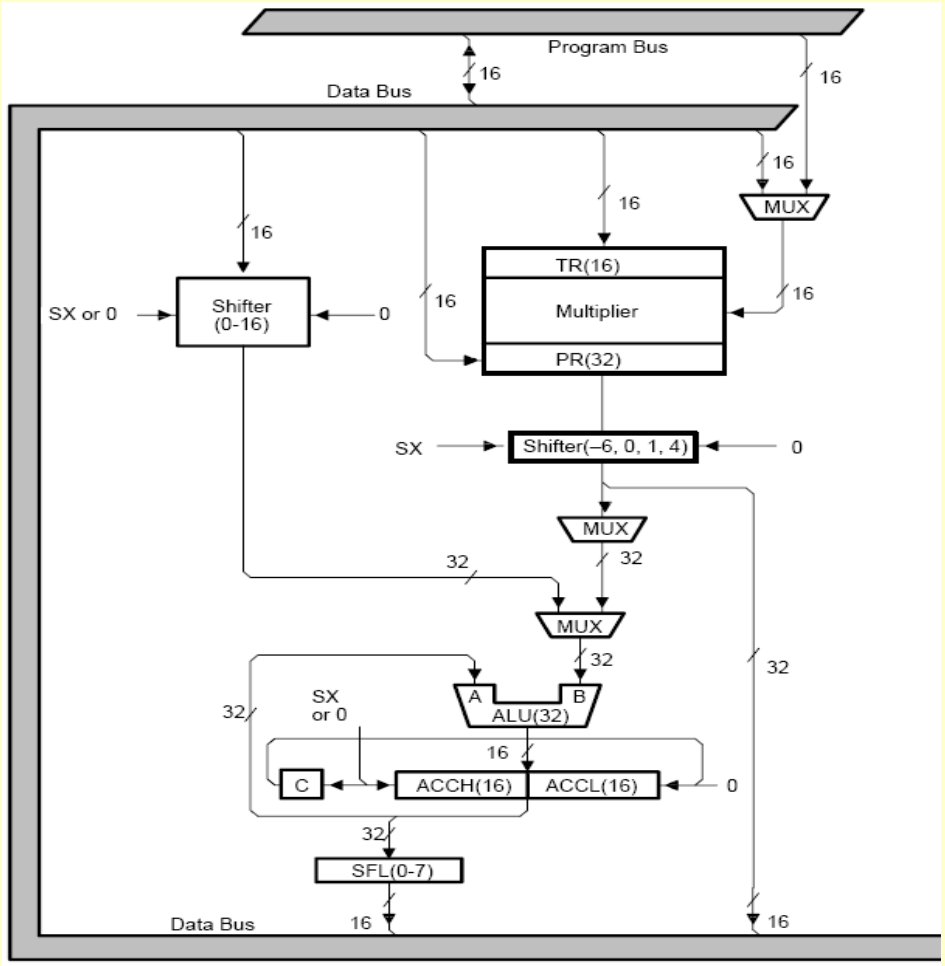


Ex.2 Write the program sequence in C2x A.L. which compute :

Y = A*X1+B*X2+C*X3+D*X4 ;

```
. data
A  word ...
B  word ...
......
X1  word
...............

ZAC                 ;ACC=0
LT  X1              ;T=X1
MPY  A              ;P=A*X1
LTA  X2             ; ACC=AX1, T=X2
MPY  B              ; P=B*X2
LTA  X3             ;ACC=ACC+BX2, T=X3
MPY  C              ;P=C*X3
LTA  X4             ;ACC=ACC+CX3, T=X4
MPY  D              ;P=D*X4
APAC                ;ACC=ACC+P ; SPM=0
SACH  Y1            ;store 32 bits result
SACL  Y2            ; at Y1,Y2
```



Ex.3 Write the program sequence in C2x A.L. which compute: Y = AX1+BX2+CX3+DX4 ;
where A, B, C, D are words in memory B2 and X1,X2,X3, X4 are elements of a string X in the block B2, using indirect addressing.
B2 adr. 60h : A,B,C,D, Y1,Y2

```
ZAC                 ;Acc=0
LARP    2           ;current AR=2
LRLK  AR2,70h       ; X string stored at adr. 70h from B2
LT A                ;T=A
MPY *+              ;P=A*X1, AR2=71h
LTA B               ; Acc=Acc+P, T=B
MPY *+              ;P=B*X2, AR2=72h
LTA C               ; Acc=Acc+P, T=C
 MPY *+             ;P=C*X3, AR2=73h
LTA  D              ; Acc=P, T=D
MPY *               ;P=D*X4
APAC                ; Acc=Acc+P
SACH  Y1            ;store 32 bits result
SACL  Y2            ; at Y1,Y2
```

Ex.4. Homework.

Block B0 :

200h	201h	202h	203h	204h	205h	206h	207h
10h	20h	40h	80h	100h	200h	400h	800h

What is the B2 (60h-67h) content after the C2x L.A. program sequence?

```
CNFP                ;B0 in memory program FF00...FFFF page 510-511
LARP 1              ;ARP=1
LRLK  AR1,60h       ;AR1=60h
LRLK  AR0,4         ;AR0=4
RPTK 7              ;RPTC=8
BLKP  FF00h,*BR0+   ;  *(AR1)=(FF00h)...x8
```

Examples.

Homework.

The B0 block contents is:

200h	201h	202h	203h	204h	205h	206h	207h
10h	20h	40h	80h	100h	200h	400h	800h

What is the contents of B2 block (60h-67h) after the sequence of program:

```
CNFP
LARP 1
LRLK  AR1,60h
LRLK  AR0,4
RPTK 7
BLKP      FF00h,*BR0+
```