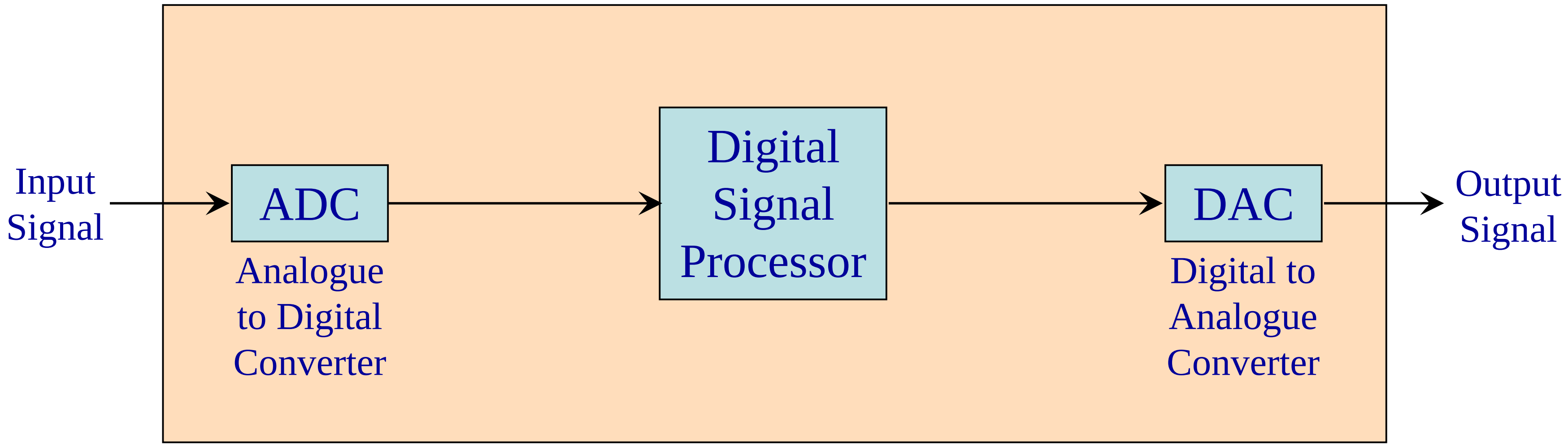
AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Procesory Sygnałowe w aplikacjach przemysłowych

Wprowadzenie

IET
Katedra Elektroniki
Kraków 2015
dr inż. Roman Rumian

Digital Signal Processing – the processing or manipulation of signals using digital techniques



DSP Devices & Architectures

- Selecting a DSP – several choices:
  - Fixed-point;
  - Floating point;
  - Application-specific devices
    (e.g. FFT processors, speech recognizers,etc.).
- Main DSP Manufacturers:
  - Texas Instruments (http://www.ti.com)
  - Motorola (http://www.freescale.com)
  - Analog Devices (http://www.analog.com)

Typical DSP Operations

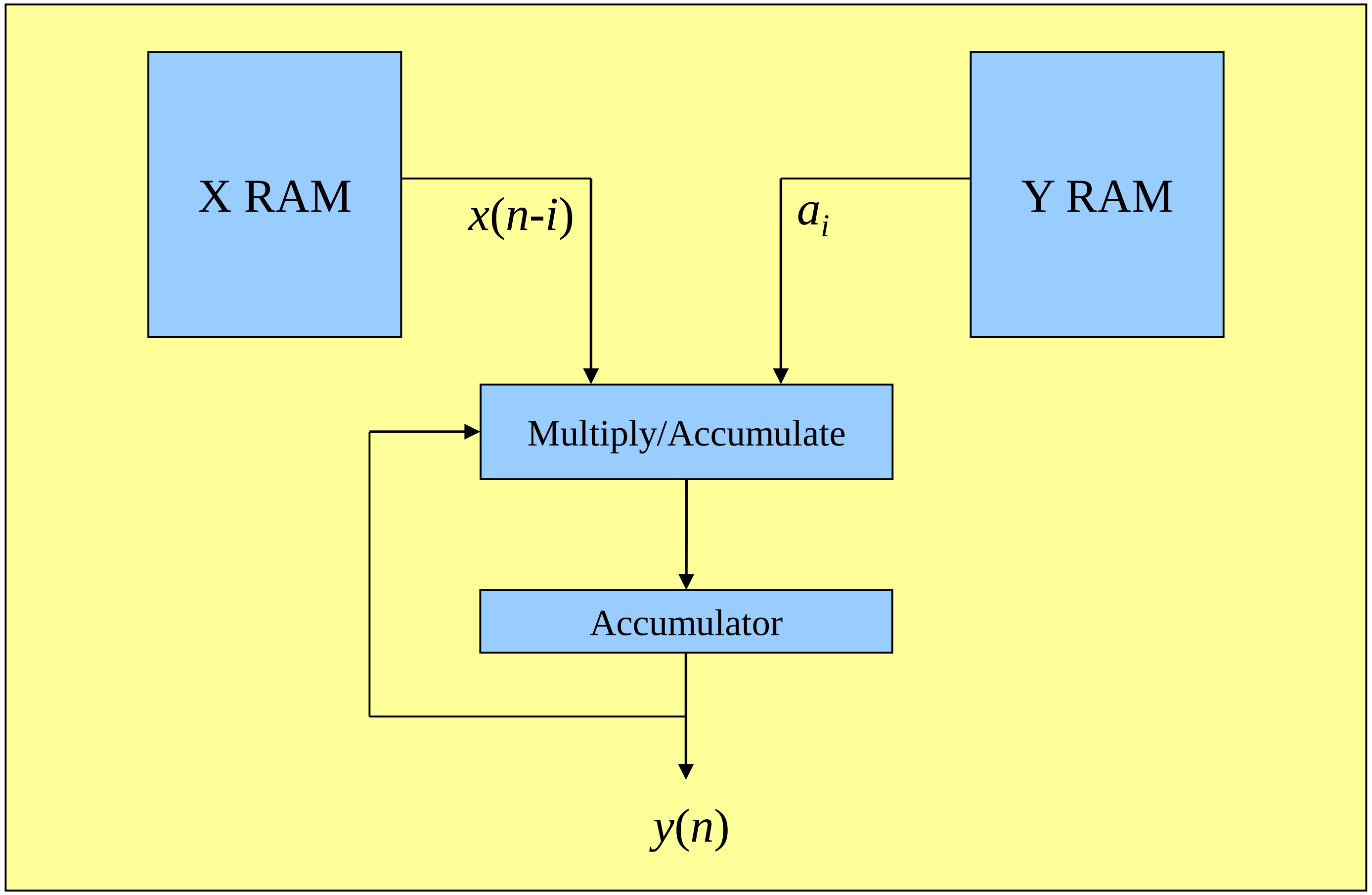
- Filtering
- Energy of Signal
- Frequency transforms

$$y(n) = \sum_{i=0}^{L-1} a_i x(n-i)$$

Pseudo C code

```
for (n=0; n<N; n++)/*block filtring*/
{
   s=0;
   for (i=0; i<L; i++)
   {
      s += a[i] * x[n-i];
   }
   y[n] = s;
}
```

Traditional DSP Architecture

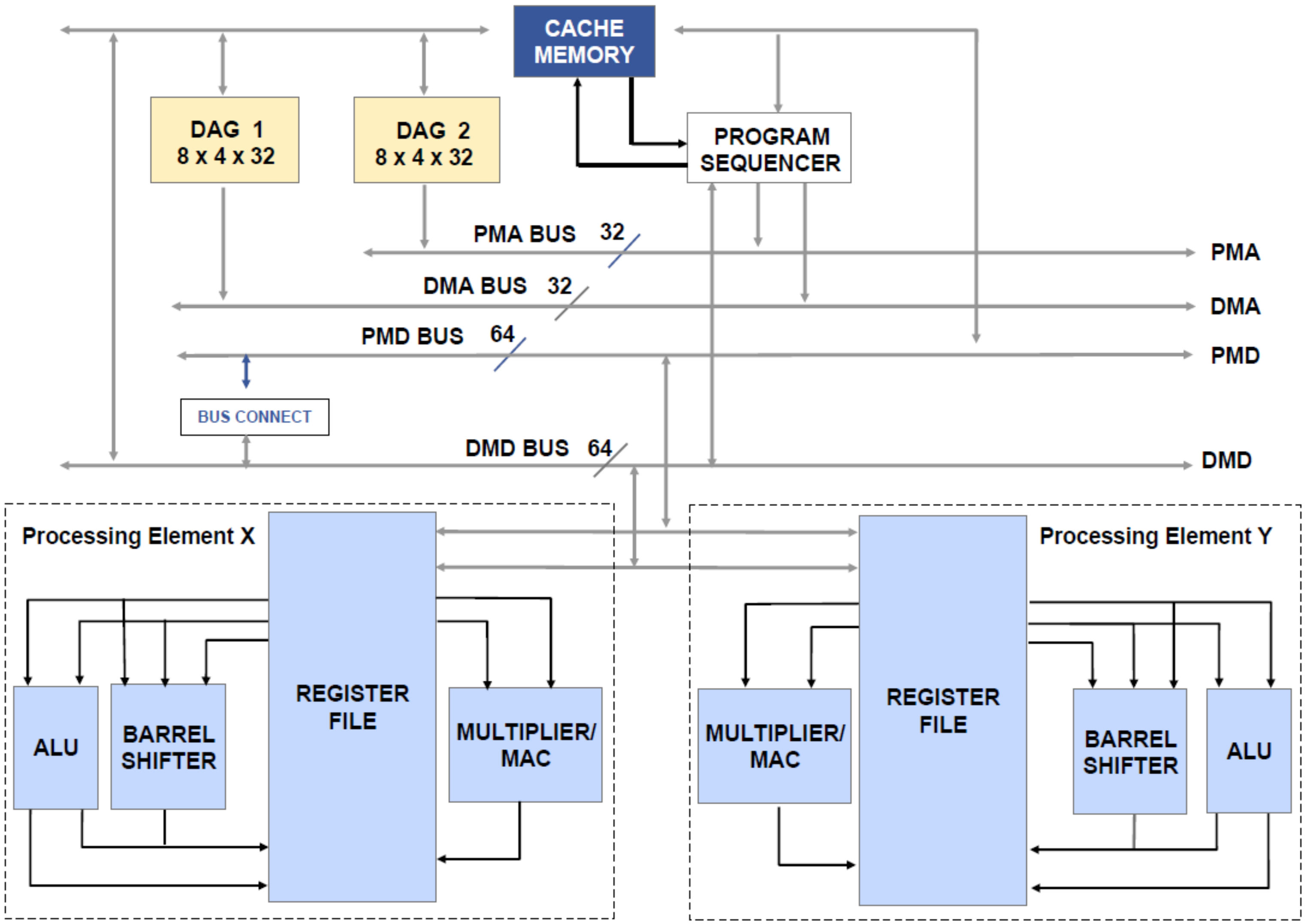


AGH

SHARC Benchmarks

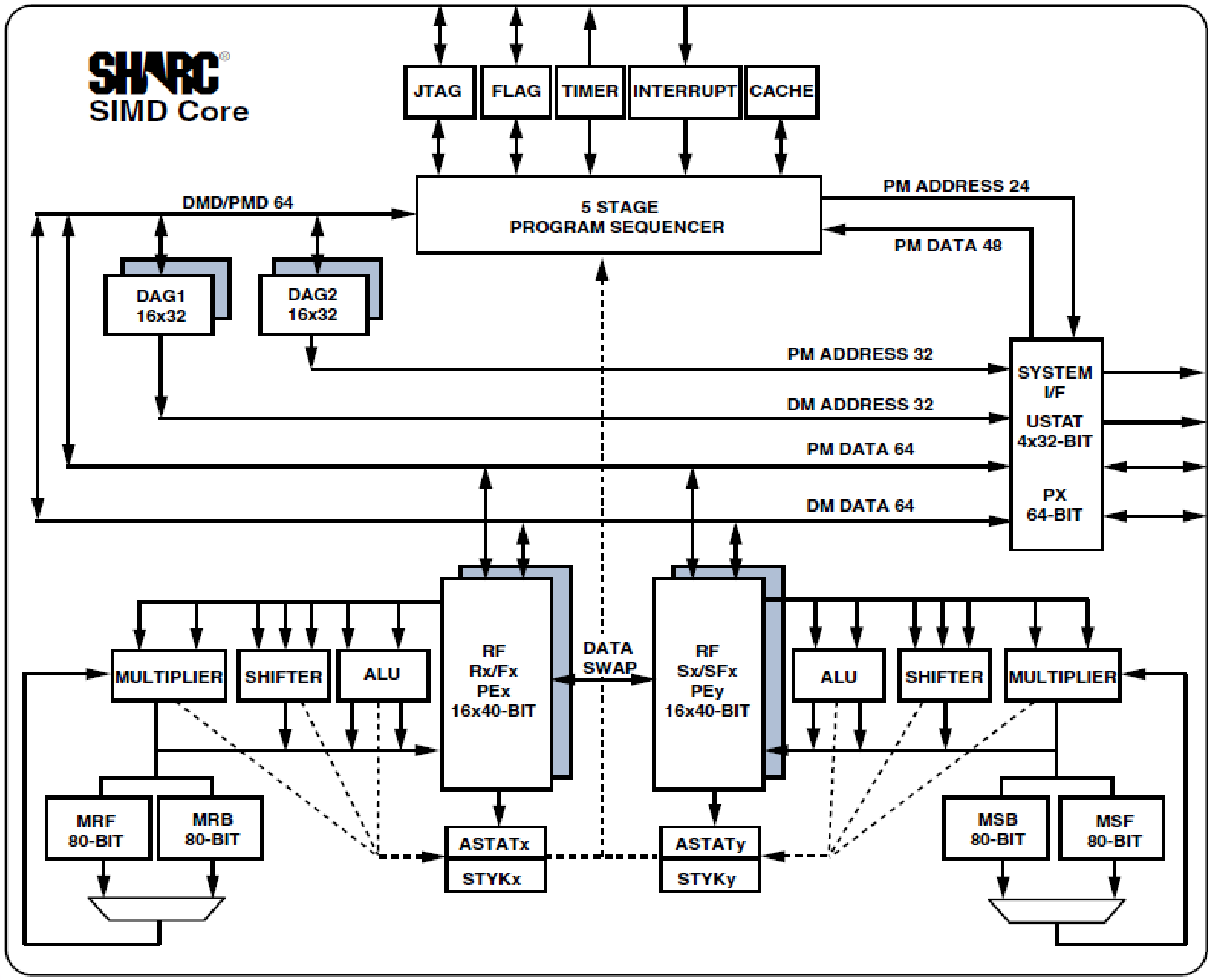
	ADSP-21160N ADSP-21161N SIMD	ADSP-21261 SIMD	ADSP-21262 ADSP-21266 SIMD	ADSP-21371 ADSP-21375 SIMD	ADSP-21364 ADSP-21365 SIMD	ADSP-21368 ADSP-21369 SIMD	ADSP-2146x SIMD
Clock Cycle	100 MHz	150 MHz	200 MHz	266 MHz	333 MHz	400 MHz	450 MHz
Instruction Cycle Time	10 ns	6.67 ns	5 ns	3.75 ns	3 ns	2.5 ns	2.22 ns
MFLOPS Sustained	400 MFLOPS	600 MFLOPS	800 MFLOPS	1064 MFLOPS	1332 MFLOPS	1600 MFLOPS	1800 MFLOPS
MFLOPS Peak	600 MFLOPS	900 MFLOPS	1200 MFLOPS	1596 MFLOPS	1998 MFLOPS	2400 MFLOPS	2700 MFLOPS
1024 Point Complex FFT (Radix 4, with bit reversal)	92 μs	61.3 μs	46 μs	34.5 μs	28 μs	23 us	20.44 μs
FIR Filter (per tap)	5 ns	3.3 ns	2.5 ns	1.88 ns	1.5 ns	1.25 ns	1.11 ns
IIR Filter (per biquad)	20 ns	13.3 ns	10 ns	7.5 ns	6 ns	5 ns	4.43 ns
Matrix Multiply (pipelined) [3x3] * [3x1] [4x4] * [4x1]	45 ns 80 ns	30 ns 53.3 ns	22.5 ns 40 ns	16.91 ns 30.07 ns	13.5 ns 24 ns	11.25 ns 20 ns	10.00 ns 17.78 ns
Divide (y/x)	30 ns	20 ns	15 ns	11.27 ns	9 ns	7.5 ns	6.67 ns
Inverse Square Root	45 ns	30 ns	22.5 ns	16.91 ns	13.5 ns	11.25 ns	10.00 ns

SHARC SIMD Core

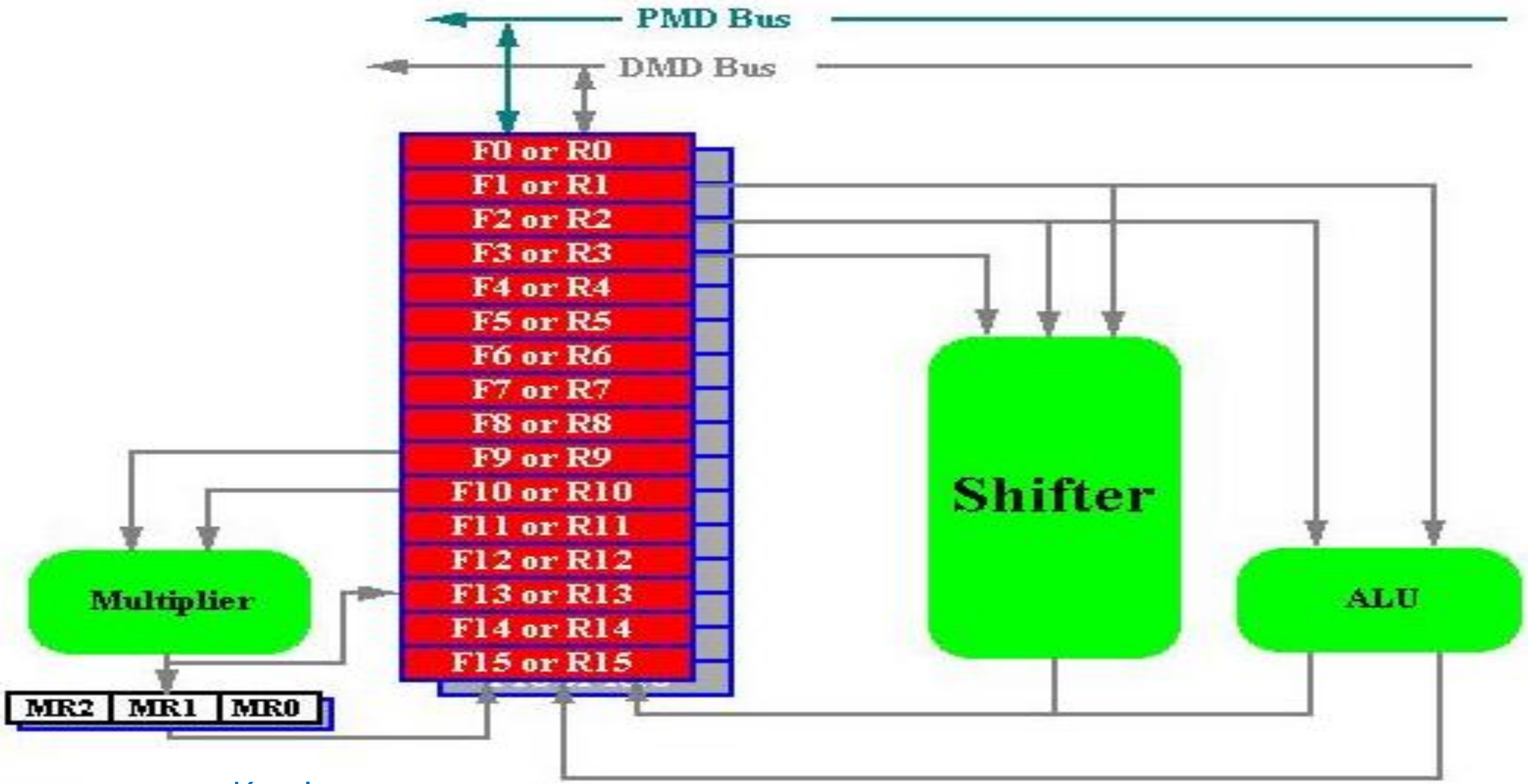


SHARC

‘S’uper ‘H’arvard ‘ARC’hitecture



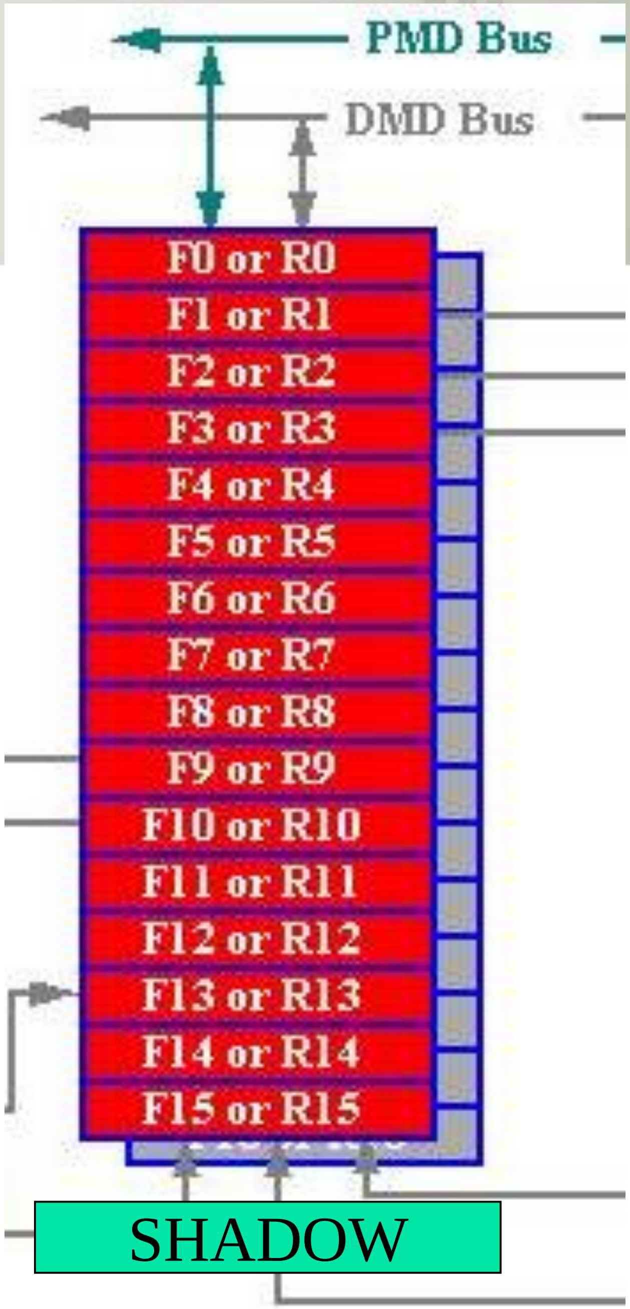
Register File and COMPUTE Units



- Key issues:
  - 5 data paths FROM COMPUTE units
  - 5 data paths TO COMPUTE units
  - Highly parallel operations UNDER THE RIGHT CONDITIONS

Register File – BIT STORAGE

- Key issues
  - 40 bits wide
  - Top 32 bits used for integer
  - Top 32 bits used for float
  - 40 bits for precision float
  - 32 registers available
    16 at a time
- A Register is always 40 bits
  - can be processed
    as a float
  - can be processed
    as an integer
  - Must convert integer<-> float



Instruction format

- Instructions are 48 bits wide
- 23 bits – COMPUTE field – are available for computer operations – See appendix B-1
- Single-function format

22 21-20 19-12 11-8 7-4 3-0

11-8 destination 7-4, 3-0 – source register

19-12 opcode associated with computation unit (bits 21-20) (ALU, MAC or SHIFTER)

Bit 22 always a 0 for single function format

SHARC assembly language

- Algebraic notation terminated by semicolon:

DAG1 and DAG2 registers

```
   R1=DM(M0,I0), R2=PM(M8,I8);!comment
label:   R3=R1+R2;
```

data memory access

program memory access

Sample ALU Instructions

ALU INSTRUCTIONS – COMPUTE OPERATION

INTEGER

Rn = Rx + Ry
Rx = Rx – Ry
Rn = Rx + Ry + CI (Carry In)
Rn = Rx - Ry + CI - 1
Rn = (Rx + Ry) / 2
COMP(Rx, Ry)
Rn = Rx + CI – 1
Rn = Rx + 1
Rn = Rx – 1
Rn = -Rx
Rn = ABS Rx
Rn = PASS Rx
Rn = Rx AND Ry
Rn = Rx OR Ry
Rn = NOT Rx
Rn = MIN(Rx, Ry)
Rn = MAX(Rx, Ry)
Rn = CLIP Rx by Ry

FLOAT

Fn = Fx + Fy
Fn = Fx - Fy
Fn = ABS(Fx + Fy)
Fn = ABS(Fx – Fy)
Fn = (Fx + Fy) / 2
COMP(Fx, Fy)
Fn = - Fx
Fn= ABS Fx
Fn= PASS Fx
Fn = RND Fx
Fn = SCALB Fx BY Ry
Rn = MANT Fx
Rn = LOGB Fx
Rn = FIX Fx *BY Ry*
Fn = FLOAT Rx *BY Ry*
Rn = TRUNC Fx
Fn = RECIPS Fx
Fn = RSQRTS Fx
Fn = Fx COPYSIGN Fy
Fn = MIN(Fx, Fy)
Fn = MAX(Fx, Fy)
Fn = CLIP Fx by Fy

MULTIFUNCTION Instructions

MULTIFUNCTION – COMPUTE OPERATION

On certain registers only, unlike standard COMPUTE
Multiplication FN = FQ * FR, with FQ=F(0,1,2,3) and FR=(4,5,6,7)
ALU Compute FN = FX op FY, FX=F(8,9,10,11),FY=(12,13,14,15)

Dual Add/Subtract – integer or float allowed
FN = FX + FY, FM = FX – FY;
Parallel Multiplier / ALU operation – integer of float allowed
FN = FQ * FR, FM = Any ALU operation using FX and FY
Parallel Multiplier with Dual Add/ Subtract
FN = FQ * FR, FM = FX + FY, FO = FX – FY;

IMMEDIATE MOVE – NOT VALID WITH *“IF COMPUTE”*

ureg ←→ dm(<addr32>);	ureg ←→ pm(<addr24>);
dm(<addr32>, Ia)←→ ureg;	pm(<addr24>, Ic) ←→ ureg;
ureg = <data32>	

Sample MULTIPLIER Instructions

MAC INSTRUCTIONS – INTEGER COMPUTE

Rn = Rx * Ry
MRB = Rx * Ry
Rn = MRB + Rx * Ry
MRB = MRB + Rx * Ry
Rn = MRB – Rx * Ry
MRB = MRB – Rx * Ry
Rn = SAT MRB
MRB = SAT MRB
Rn = RND MRB
MRB = RND MRB
Rn = MR

MRF = Rx * Ry
Rn = MRF + Rx * Ry
MRF = MRF + Rx * Ry
Rn = MRF – Rx * Ry
MRF = MRF – Rx * Ry
Rn = SAT MRF
MRF = SAT MRF
Rn = RND MRF
MRF = RND MRF
MR = Rn

MAC INSTRUCTIONS -- FLOAT COMPUTE

Fn = Fx * Fy

Sample SHIFTER Instructions

SHIFTER OPERATIONS

Rn = LSHIFT Rx BY Ry/<data8>

Rn = Rn OR LSHIFT Rx BY Ry/<data8>

Rn = ASHIFT Rx BY Ry/<data8>

Rn = ROT Rx BY Ry/<data8>

Rn = BCLR Rx BY Ry/<data8>

Rn = BSET Rx BY Ry/<data8>

Rn = BTGL Rx BY Rx/<data8>

BTST Rx BY Ry/<data8>

Rn = *Rn OR* FDEP Rx BY Ry/<bit6>:<len6> *(SE)*

Rn = Rx BY Ry/<bit 6>:<len6> *(SE)*

Rn = EXP Rx *(EX)*

Rn = LEFTZ Rx

Rn = LEFT0 Rx

Rn = FPACK Fx

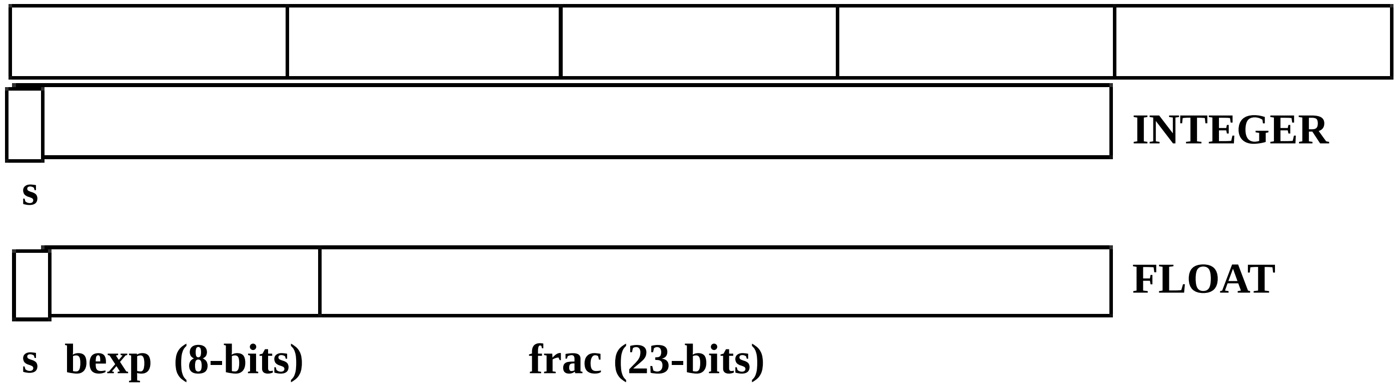
Fn = UNPACK Rx

MISCELLANEOUS Instructions

MISCELLANEOUS

BIT SET/CLR/TGL/TST,XOR srg <data32>;
<data32> = MASK for SET/CLR/TGL/TST
MODIFY (Ia, <data32>)/(Ic, <data24>)
BITREV (Ia, <data32>)/(Ic, <data24>)
PUSH/POP LOOP, PUSH/POP STS PUSH/POP PCSTK,
FLUSH CACHE;
NOP; IDLE; IDLE16 – *idle till interrupt – low power*

SHARC NUMBER FORMATS
STANDARD SHRC REGISTERS ARE 40-bits wide



UNIVERSAL REGISTER INFORMATION – UREG

DATA – *R0 to R15* INDEX (Address) – *I0 to I7, I8 to I15*
MODIFY -- *M0 to M7 M8 to M15* LENGTH – *L0 to L7, L8 to L15*
BASE – *B0 to B7, B8 to B15* *(Setting Bx also set Ix)*
Ia/Mb refers to DAG1 (dm) registers – Ic/Md refers to DAG2 (pm)

PROGRAM SEQUENCER – *PC, PCSTK, PCSTKP, FADDR, DADDR. LADDR. CURLCNTR, LCNTR*
BUS EXCHANGE -- *PX1, PX2, PX*
TIMER – *TPERIOD, TCOUNT*
SYSTEM REGISTERS – sreg -- *MODE1, MODE2, IRPTL, IMASK, IMASKP, ASTAT, STKY, USTAT1, USTAT2*

Addresing Modes

PRE AND POST MODIFY OPERATIONS

PRE-MOD -- (Mb, Ia) – Use address (Mb + Ia) Leave Ia unchanged

IF La register = 0 – causes normal array operation

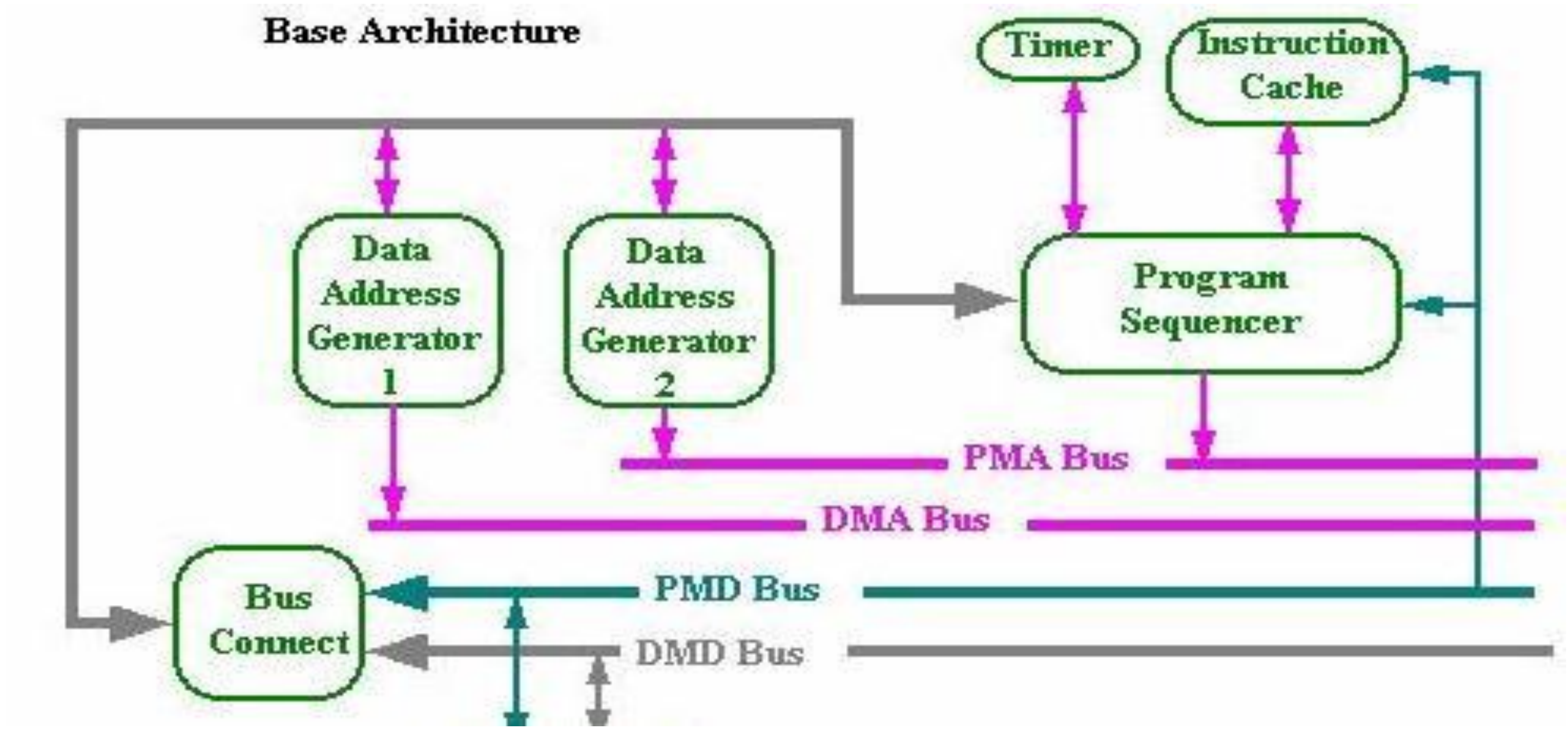
POST-MOD -- (Ia. Mb) – Use address (Ia) – Change Ia to Ia + Mb

IF La register != 0 – causes circular buffer operations

POST-MOD – (Ia, Mb) – Use address (Ia) – Change Ia to Ia + Mb

then perform *Ia –La or Ia + La* until *Ia in range Ba to Ba + La –1*

Memory Accesses



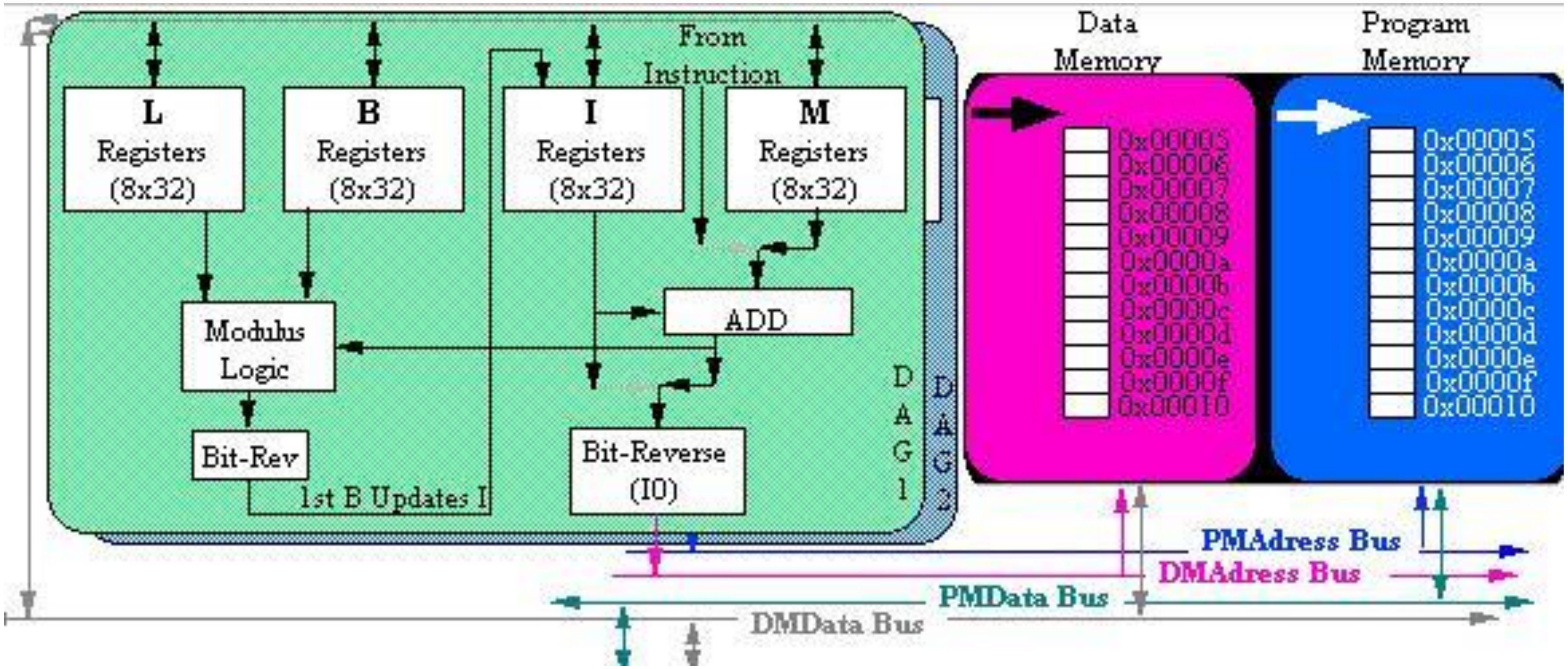
Under the right conditions -- 3 memory accesses at same time

Program Memory, Data Memory, Instruction Cache

PLUS up to 2 ALU + 1 MAC operations at the same time

PLUS background DMA activity

Data Address Generators -- DAG



There is only 1 memory, but it is broken into 2 sections

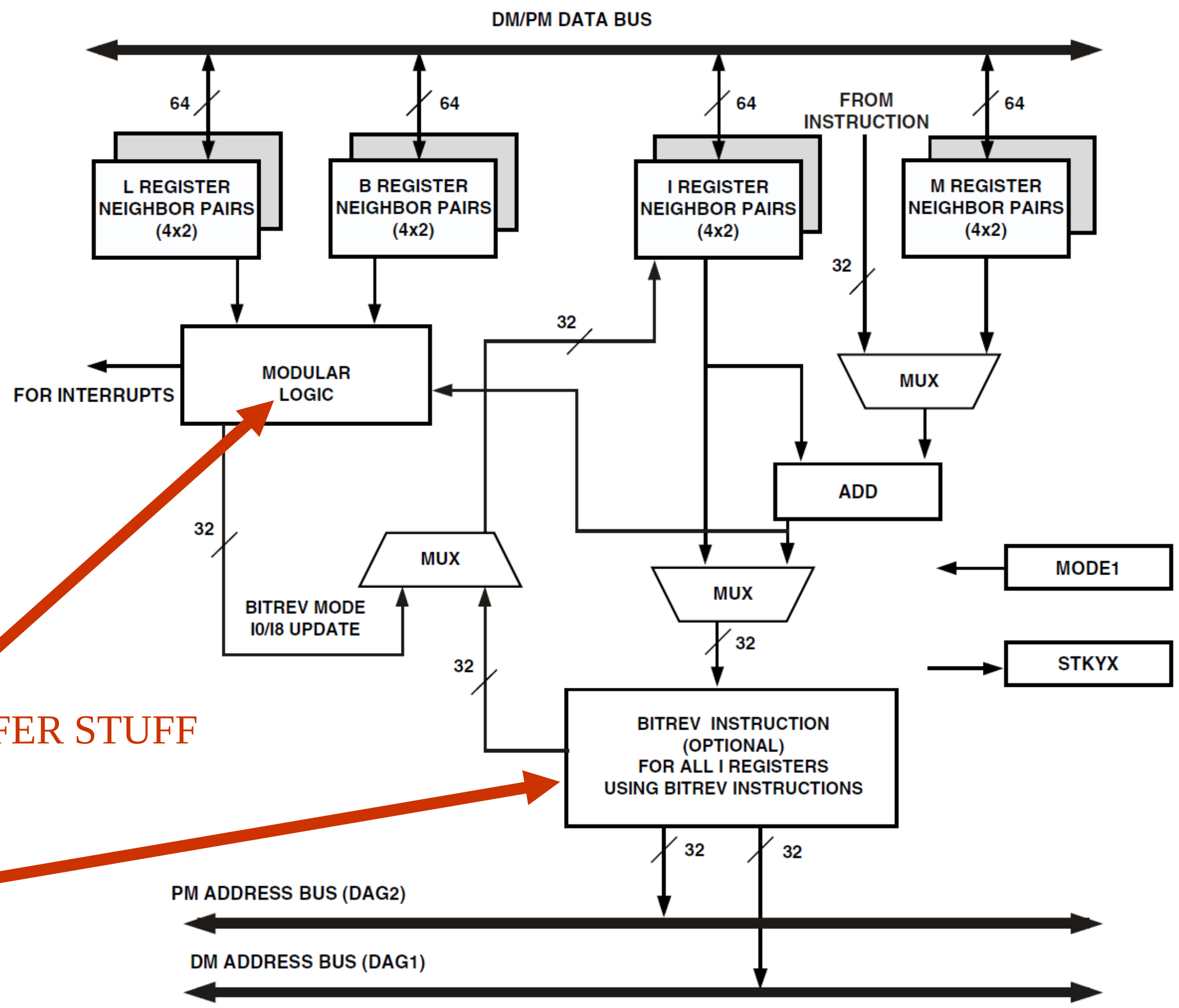
DAG1 -- best for accessing Data memory section (0 -- 7)

DAG2 -- best for accessing Program memory section (8 -- 15)

MUST be used in this fashion for simultaneous memory ops

Also an alternate set of DAGs (SHADOW DAGs)

Register and Register Ops in DAG1


DM/PM DATA BUS
64
64
64
64
FROM INSTRUCTION
L REGISTER NEIGHBOR PAIRS (4x2)
B REGISTER NEIGHBOR PAIRS (4x2)
I REGISTER NEIGHBOR PAIRS (4x2)
M REGISTER NEIGHBOR PAIRS (4x2)
32
32
MODULAR LOGIC
FOR INTERRUPTS
MUX
ADD
32
MUX
BITREV MODE I0/I8 UPDATE
MUX
MODE1
STKYX
32
32
BITREV INSTRUCTION (OPTIONAL) FOR ALL I REGISTERS USING BITREV INSTRUCTIONS
32
32
PM ADDRESS BUS (DAG2)
DM ADDRESS BUS (DAG1)

SPECIAL CIRCBUFFER STUFF

SPECIAL FFT BIT

DAG register info

- Index registers
  - I0 -- I7 (dm -- data mem), I8 -- I15 (pm -- program mem)
  - "like" 68K address registers A0 -- A6
- Modify registers M0 -- M7, M8 -- M15
  - Can be offset registers (*c.f 68K (4, SP)*
  - Can be used for high speed post increment
- Special Hardware for Circular Buffers
  - Base registers B0 -- B7, B8 -- B15
  - Length registers L0 -- L7, L8 -- L15
  - *See labs 2 -- 4 and associated lectures*

Address Versus Word Size

The processor’s internal memory accommodates the following word sizes:

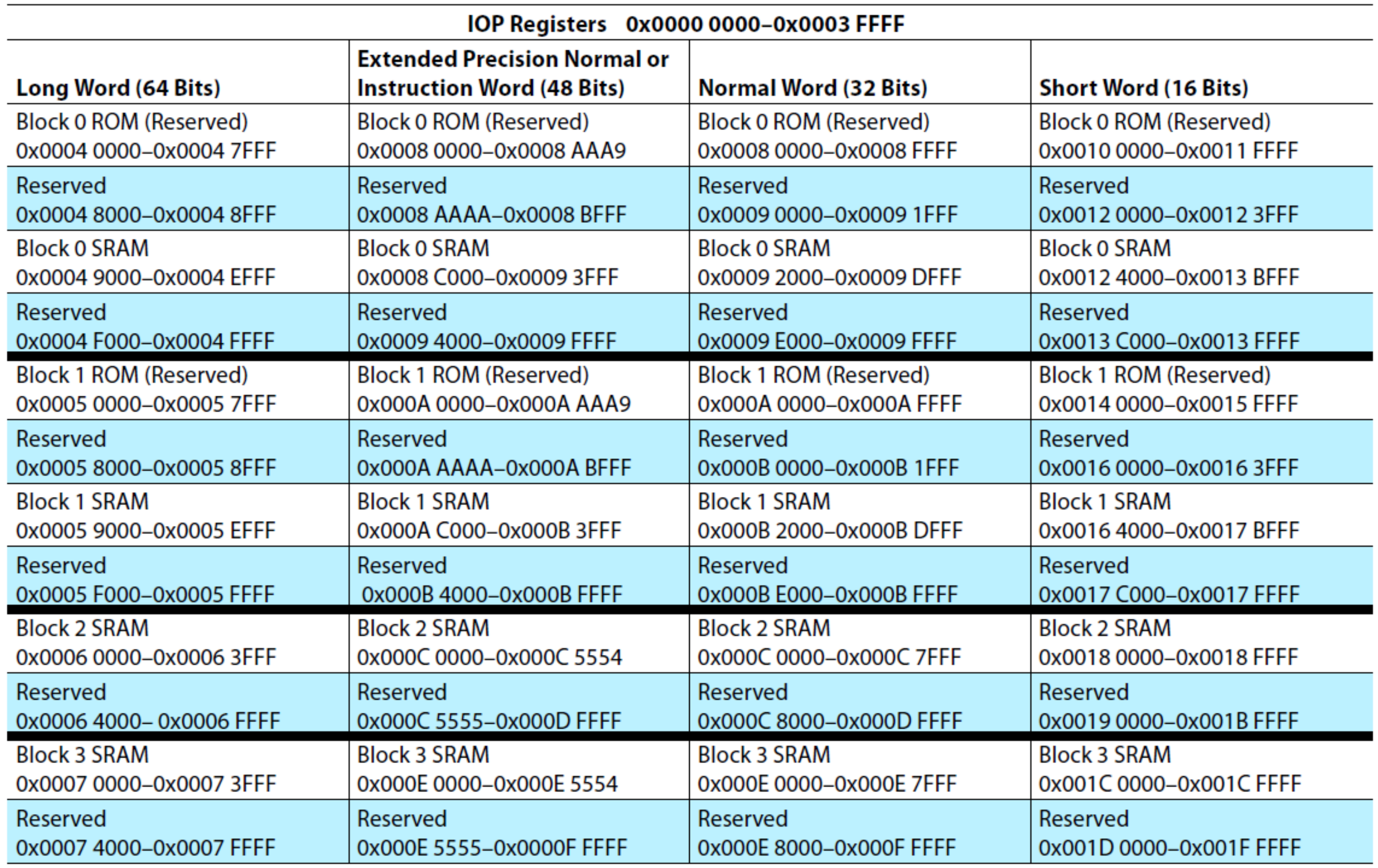
- 64-bit long word data (LW)
- 40-bit extended-precision normal word data (NW, 48-bit)
- 32-bit normal word data (NW, 32-bit)
- 16-bit short word data (SW, 16-bit)

ⓘ **Only the address space determines which memory word size is accessed.** An important item to note is that the DAG automatically adjusts the output address per the word size of the address location (short word, normal word, or long word). This address adjustment allows internal memory to use the address directly as shown in the following example.

```
I15=LW_addr;
pm(i15,0)=r0; /* 64-bit transfer */

I7=NW_addr;
dm(i7,0)=r8; /* 32-bit transfer */

I7=SW_addr;
dm(i7,0)=r14; /* 16-bit transfer */
```

Internal memory map

Table 4. Internal Memory Space (5 MBits—ADSP-21486/ADSP-21487/ADSP-21489)[1]



IOP Registers 0x0000 0000–0x0003 FFFF			
Long Word (64 Bits)	**Extended Precision Normal or Instruction Word (48 Bits)**	**Normal Word (32 Bits)**	**Short Word (16 Bits)**
Block 0 ROM (Reserved) 0x0004 0000–0x0004 7FFF	Block 0 ROM (Reserved) 0x0008 0000–0x0008 AAA9	Block 0 ROM (Reserved) 0x0008 0000–0x0008 FFFF	Block 0 ROM (Reserved) 0x0010 0000–0x0011 FFFF
Reserved 0x0004 8000–0x0004 8FFF	Reserved 0x0008 AAAA–0x0008 BFFF	Reserved 0x0009 0000–0x0009 1FFF	Reserved 0x0012 0000–0x0012 3FFF
Block 0 SRAM 0x0004 9000–0x0004 EFFF	Block 0 SRAM 0x0008 C000–0x0009 3FFF	Block 0 SRAM 0x0009 2000–0x0009 DFFF	Block 0 SRAM 0x0012 4000–0x0013 BFFF
Reserved 0x0004 F000–0x0004 FFFF	Reserved 0x0009 4000–0x0009 FFFF	Reserved 0x0009 E000–0x0009 FFFF	Reserved 0x0013 C000–0x0013 FFFF
Block 1 ROM (Reserved) 0x0005 0000–0x0005 7FFF	Block 1 ROM (Reserved) 0x000A 0000–0x000A AAA9	Block 1 ROM (Reserved) 0x000A 0000–0x000A FFFF	Block 1 ROM (Reserved) 0x0014 0000–0x0015 FFFF
Reserved 0x0005 8000–0x0005 8FFF	Reserved 0x000A AAAA–0x000A BFFF	Reserved 0x000B 0000–0x000B 1FFF	Reserved 0x0016 0000–0x0016 3FFF
Block 1 SRAM 0x0005 9000–0x0005 EFFF	Block 1 SRAM 0x000A C000–0x000B 3FFF	Block 1 SRAM 0x000B 2000–0x000B DFFF	Block 1 SRAM 0x0016 4000–0x0017 BFFF
Reserved 0x0005 F000–0x0005 FFFF	Reserved 0x000B 4000–0x000B FFFF	Reserved 0x000B E000–0x000B FFFF	Reserved 0x0017 C000–0x0017 FFFF
Block 2 SRAM 0x0006 0000–0x0006 3FFF	Block 2 SRAM 0x000C 0000–0x000C 5554	Block 2 SRAM 0x000C 0000–0x000C 7FFF	Block 2 SRAM 0x0018 0000–0x0018 FFFF
Reserved 0x0006 4000– 0x0006 FFFF	Reserved 0x000C 5555–0x000D FFFF	Reserved 0x000C 8000–0x000D FFFF	Reserved 0x0019 0000–0x001B FFFF
Block 3 SRAM 0x0007 0000–0x0007 3FFF	Block 3 SRAM 0x000E 0000–0x000E 5554	Block 3 SRAM 0x000E 0000–0x000E 7FFF	Block 3 SRAM 0x001C 0000–0x001C FFFF
Reserved 0x0007 4000–0x0007 FFFF	Reserved 0x000E 5555–0x0000F FFFF	Reserved 0x000E 8000–0x000F FFFF	Reserved 0x001D 0000–0x001F FFFF

INSTRUCTIONS AND DELAY JUMP SLOT

```
        R2 = 1;
        R8 = pass R2;
        If NE jump(pc, _LABEL) (DB);
                R8 = 2;             Execute whether jump or not
                R7 = 1;             Execute whether jump or not
        R8 = 3;
_LABEL:
```

Warning: R7 = 1, whether jump OR NOT,
R8 = 3 if jump DOES NOT OCCUR,
R8 = 2 if jump occurs and not 1

COMPUTE AND MOVE INSTRUCTIONS (PARALLEL)

compute, dm(Ia, Mb) ←→ dreg1, pm(Ic, Md) ←→ dreg2;

IF condition compute; ***N.B.*** *italics = optional part of instruction*

N.B. ***“****IF” operation affects the WHOLE instruction*

IF condition compute, dm(Pre/Post with MREGISTERS)←→ureg;

IF condition compute, pm(Pre/Post with MREGISTERS)←→ureg;

N.B. *ureg can't be from same DAG as Pre/Post registers*

IF condition compute, dm(Pre/Post with <data6>) ←→ ureg;

IF condition compute, pm(Pre/Post with <data6>) ←→ ureg;

IF condition compute, dreg←→dm(IREG, MREGISTER)

IF condition compute, dreg←→pm(IREG, MREGISTER)

IF condition compute, reg1 = ureg2;

IF condition shiftimm, dm(IREG, MREGISTER)←→dreg;

IF condition shiftimm, pm(IREG, MREGISTER)←→dreg;

IF condition compute, MODIFY (IREG, MREGISTER);

PROGRAM FLOW CONTROL (PARALLEL)

N.B. *italics = optional part of instruction*

N.B. *"IF" operation affects the WHOLE instruction*

IF condition JUMP <addr24> (DB/LA/CI);
IF condition JUMP (PC, <reladdr24>) (DB/LA/CI);
IF condition CALL <addr24> (DB);
IF condition CALL (PC, <reladdr24> (DB);
IF condition JUMP (Md, Ic) (DB/LA/CI), *compute;*
IF condition JUMP (PC, <reladdr24>) (DB/LA/CI) *ELSE compute*;
IF condition CALL (Md, Ic) (DB/LA/CI), *compute;*
IF condition CALL (PC, <reladdr24>) (DB/LA/CI) *ELSE compute*;
IF condition JUMP (Md. Ic), ELSE *compute*, dm(Ia, Mb) = dreg;
IF condition JUMP (Md. Ic), ELSE *compute*, dreg = dm(Ia, Ib);
IF condition RTS (DB/LR), *compute*;
IF condition RTS (DB/LR), *ELSE compute*;
IF condition RTI (DB/LR), *compute*;
IF condition RTI (DB/LR), *ELSE compute*;
LCNTR = <data16>, DO <addr24> UNTIL LCE;
LCNTR = ureg,   DO <PC, <reladdr24> UNTIL LCE;
General Form of DO   DO <addr24> UNTIL termination;
General Form of DO   DO (PC, <reladdr24>) until termination

(DB) Delayed branch
(LA) Loop abort (pop loop and PC stacks on branch)
(CI) Clear interrupt

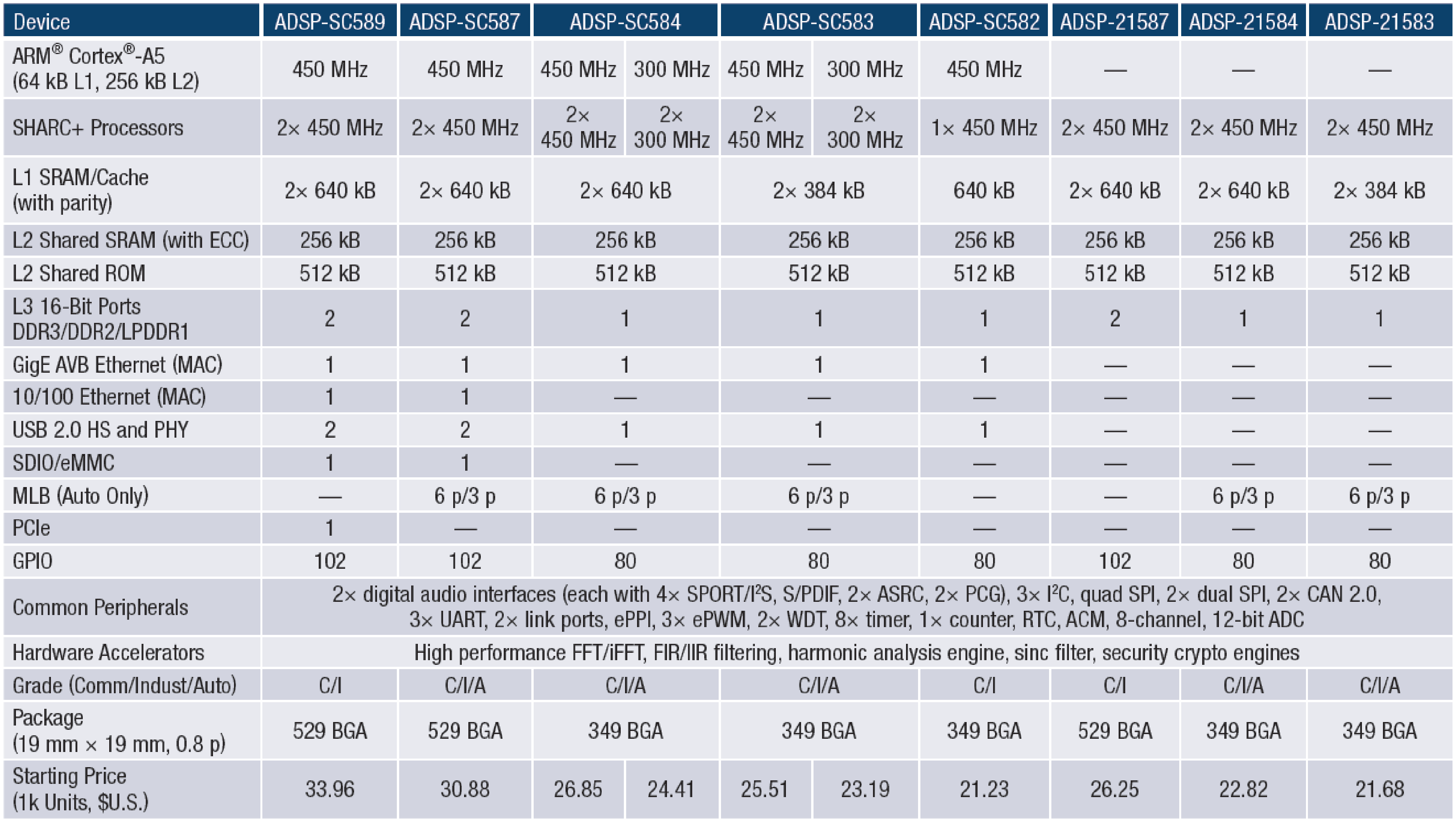
Instrukcje skoków bezwarunkowych i warunkowych

```
JUMP etykieta //skok bezwarunkowy

CALL etykieta //bezwarunkowe wywołanie podprogramu



IF NE JUMP etykieta //skok warunkowy

IF AC CALL etykieta //warunkowe wywołanie podprogramu
```

Niektóre pozostałe instrukcje także mogą być wykonywane warunkowo, np. :

```
IF EQ DM(I0,M0) = R2;
IF EQ R8 = R2;
```

ADSP-SC58x & ADSP-2158x Family Comparison



Device	ADSP-SC589	ADSP-SC587	ADSP-SC584		ADSP-SC583		ADSP-SC582	ADSP-21587	ADSP-21584	ADSP-21583
ARM® Cortex®-A5 (64 kB L1, 256 kB L2)	450 MHz	450 MHz	450 MHz	300 MHz	450 MHz	300 MHz	450 MHz	—	—	—
SHARC+ Processors	2× 450 MHz	2× 450 MHz	2× 450 MHz	2× 300 MHz	2× 450 MHz	2× 300 MHz	1× 450 MHz	2× 450 MHz	2× 450 MHz	2× 450 MHz
L1 SRAM/Cache (with parity)	2× 640 kB	2× 640 kB	2× 640 kB		2× 384 kB		640 kB	2× 640 kB	2× 640 kB	2× 384 kB
L2 Shared SRAM (with ECC)	256 kB	256 kB	256 kB		256 kB		256 kB	256 kB	256 kB	256 kB
L2 Shared ROM	512 kB	512 kB	512 kB		512 kB		512 kB	512 kB	512 kB	512 kB
L3 16-Bit Ports DDR3/DDR2/LPDDR1	2	2	1		1		1	2	1	1
GigE AVB Ethernet (MAC)	1	1	1		1		1	—	—	—
10/100 Ethernet (MAC)	1	1	—		—		—	—	—	—
USB 2.0 HS and PHY	2	2	1		1		1	—	—	—
SDIO/eMMC	1	1	—		—		—	—	—	—
MLB (Auto Only)	—	6 p/3 p	6 p/3 p		6 p/3 p		—	—	6 p/3 p	6 p/3 p
PCIe	1	—	—		—		—	—	—	—
GPIO	102	102	80		80		80	102	80	80
Common Peripherals	2× digital audio interfaces (each with 4× SPORT/I^{2}S, S/PDIF, 2× ASRC, 2× PCG), 3× I^{2}C, quad SPI, 2× dual SPI, 2× CAN 2.0, 3× UART, 2× link ports, ePPI, 3× ePWM, 2× WDT, 8× timer, 1× counter, RTC, ACM, 8-channel, 12-bit ADC									
Hardware Accelerators	High performance FFT/iFFT, FIR/IIR filtering, harmonic analysis engine, sinc filter, security crypto engines									
Grade (Comm/Indust/Auto)	C/I	C/I/A	C/I/A		C/I/A		C/I	C/I	C/I/A	C/I/A
Package (19 mm × 19 mm, 0.8 p)	529 BGA	529 BGA	349 BGA		349 BGA		349 BGA	529 BGA	349 BGA	349 BGA
Starting Price (1k Units, $U.S.)	33.96	30.88	26.85	24.41	25.51	23.19	21.23	26.25	22.82	21.68

AGH