SHARC instruction set

- SHARC programming model.
- SHARC assembly language.
- SHARC memory organization.
- SHARC data operations.
- SHARC flow of control.

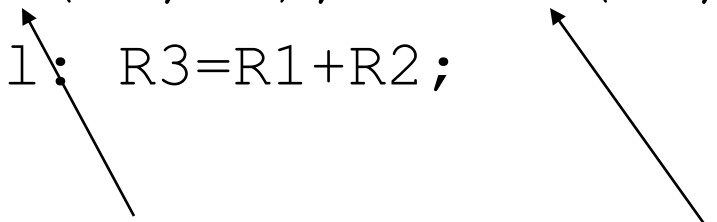
SHARC programming model

- Register files:
  - R0-R15 (aliased as F0-F15 for floating point)
- Status registers.
- Loop registers.
- Data address generator registers.
- Interrupt registers.

SHARC assembly language

- Algebraic notation terminated by semicolon:

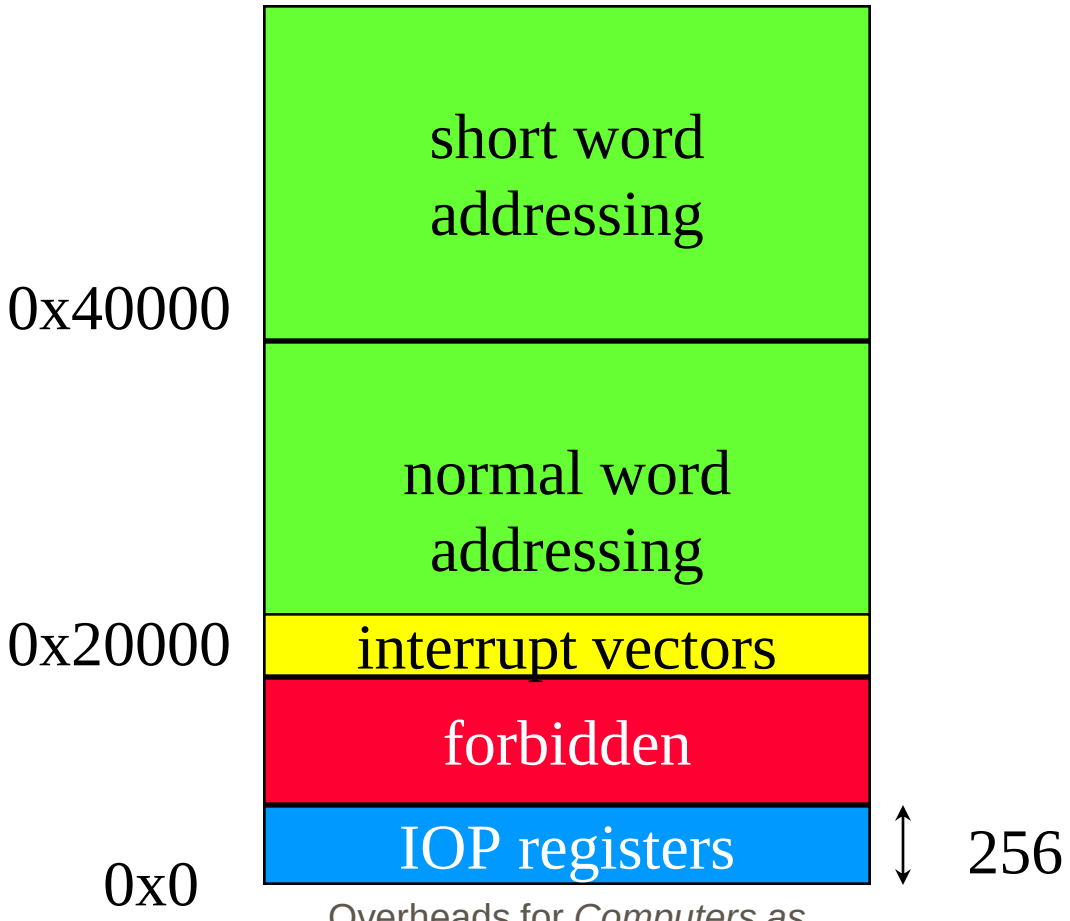
```
R1=DM(M0,I0), R2=PM(M8,I8); ! comment
label: R3=R1+R2;
```



data memory access

program memory access

SHARC memory space



SHARC data types

- 32-bit IEEE single-precision floating-point.
- 40-bit IEEE extended-precision floating-point.
- 32-bit integers.
- Memory organized internally as 32-bit words.

SHARC microarchitecture

- Modified Harvard architecture.
  - Program memory can be used to store some data.
- Register file connects to:
  - multiplier
  - shifter;
  - ALU.

SHARC mode registers

- Most important:
  - ASTAT: arithmetic status.
  - STKY: sticky.
  - MODE 1: mode 1.

Rounding and saturation

- Floating-point can be:
  - rounded toward zero;
  - rounded toward nearest.
- ALU supports saturation arithmetic (ALUSAT bit in MODE1).
  - Overflow results in max value, not rollover.

Multiplier

Fixed-point operations can accumulate into local MR registers or be written to register file. Fixed-point result is 80 bits.

Floating-point results always go to register file.

Status bits: negative, under/overflow, invalid, fixed-point undeflow, floating-point unerflow, floating-point invalid.

ALU/shifter status flags

ALU:

- zero, overflow, negative, fixed-point carry, inputsign, floating-point invalid, last op was floating-point, compare accumulation registers, floating-point under/oveflow, fixed-point overflow, floating-point invalid

Shifter:

- zero, overflow, sign

Flag operations

- All ALU operations set AZ (zero), AN (negative), AV (overflow), AC (fixed-point carry), AI (floating-point invalid) bits in ASTAT.
- STKY is sticky version of some ASTAT bits.

Example: data operations

- Fixed-point -1 + 1 = 0:
  - AZ = 1, AU = 0, AN = 0, AV = 0, AC = 1, AI = 0.
  - STKY bit AOS (fixed point underflow) not set.
- Fixed-point -2*3:
  - MN = 1, MV = 0, MU = 1, MI = 0.
  - Four STKY bits, none of them set.
- LSHIFT 0x7fffffff BY 3: SZ=0,SV=1,SS=0.

Multifunction computations

Can issue some computations in parallel:

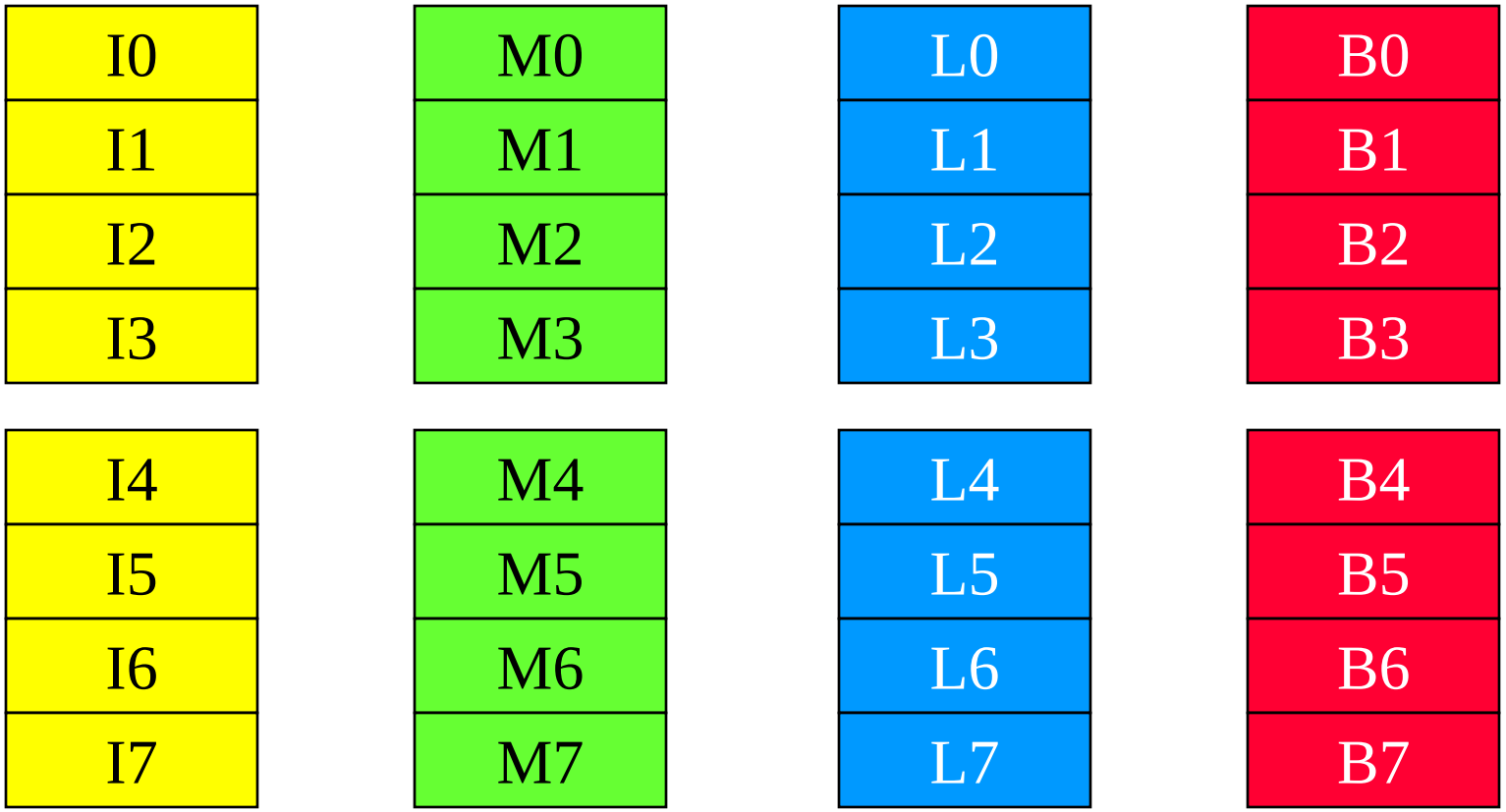
- dual add-subtract;
- fixed-point multiply/accumulate and add,subtract,average
- floating-point multiply and ALU operation
- multiplication and dual add/subtract

Multiplier operand from R0-R7, ALU operand from R8-R15.

SHARC load/store

- Load/store architecture: no memory-direct operations.
- Two data address generators (DAGs):
  - program memory;
  - data memory.
- Must set up DAG registers to control loads/stores.

DAG1 registers



Data address generators

Provide indexed, modulo, bit-reverse indexing.

MODE1 bits determine whether primary or alternate registers are active.

BASIC addressing

- Immediate value:

```
R0 = DM(0x20000000);
```

- Direct load:

```
R0 = DM(_a); ! Loads contents of _a
```

- Direct store:

```
DM(_a)= R0; ! Stores R0 at _a
```

Post-modify with update

- I register holds base address.
- M register/immediate holds modifier value.

```
R0 = DM(I3,M3) ! Load
DM(I2,1) = R1 ! Store
```

- Circular buffer: L register is buffer start index, B is buffer base address.

Data in program memory

- Can put data in program memory to read two values per cycle:

```
F0 = DM(M0,I0), F1 = PM(M8,I9);
```

- Compiler allows programmer to control which memory values are stored in.

Example: C assignments

- C:

```
x = (a + b) - c;
```

- Assembler:

```
R0 = DM(_a) ! Load a
R1 = DM(_b); ! Load b
R3 = R0 + R1;
R2 = DM(_c); ! Load c
R3 = R3-R2;
DM(_x) = R3; ! Store result in x
```

Example, cont'd.

- C:

```
y = a*(b+c);
```

- Assembler:

```
R1 = DM(_b) ! Load b
R2 = DM(_c); ! Load c
R2 = R1 + R2;
R0 = DM(_a); ! Load a
R2 = R2*R0;
DM(_y) = R23; ! Store result in y
```

Example, cont'd.

- Shorter version using pointers:

```
! Load b, c
  R2=DM(I1,M5), R1=PM(I8,M13);
  R0 = R2+R1, R12=DM(I0,M5);
  R6 = R12*R0(SSI);
  DM(I0,M5)=R8; ! Store in y
```

Example, cont'd.

- C:

```
z = (a << 2) |  (b & 15);
```

- Assembler:

```
R0=DM(_a); ! Load a
R0=LSHIFT R0 by #2; ! Left shift
R1=DM(_b); R3=#15; ! Load immediate
R1=R1 AND R3;
R0 = R1 OR R0;
DM(_z) = R0;
```

SHARC program sequencer

Features:

- instruction cache;
- PC stack;
- status registers;
- loop logic;
- data address generator;

Conditional instructions

Instructions may be executed conditionally.

Conditions come from:

- arithmetic status (ASTAT);
- mode control 1 (MODE1);
- flag inputs;
- loop counter.

SHARC jump

- Unconditional flow of control change:

```
JUMP foo
```

- Three addressing modes:
  - direct;
  - indirect;
  - PC-relative.

Branches

Types: CALL, JUMP, RTS, RTI.

Can be conditional.

Address can be direct, indirect, PC-relative.

Can be delayed or non-delayed.

JUMP causes automatic loop abort.

Example: C if statement

- C:

```
if (a > b) { x = 5; y = c + d; }
  else x = c - d;
```

- Assembler:

```
! Test
  R0 = DM(_a); R1 = DM(_b);
  COMP(R0,R1); ! Compare
  IF GE JUMP fblock;
```

C if statement, cont'd.

```
! True block
tblock: R0 = 5; ! Get value for x
  DM(_x) = R0;
  R0 = DM(_c); R1 = DM(_d);
  R1 = R0+R1;
  DM(_y)=R1;
  JUMP other; ! Skip false block
```

C if statement, cont'd.

```
! False block
fblock: R0 = DM(_c);
  R1 = DM(_d);
  R1 = R0-R1;
  DM(_x) = R1;
other:  ! Code after if
```

Fancy if implementation

- C:

```
if (a>b) y = c-d; else y = c+d;
```

- Use parallelism to speed it up---compute both cases, then choose which one to store.

Fancy if implementation, cont'd.

```
! Load values
  R1=DM(_a); R2=DM(_b);
  R3=DM(_c); R4=DM(_d);
! Compute both sum and difference
  R12 = r2+r4, r0 = r2-r4;
! Choose which one to save
  comp(r8,r1);
  if ge r0=r12;
  dm(_y) = r0 ! Write to y
```

DO UNTIL loops

DO UNTIL instruction provides efficient looping:

```
        LCNTR=30, DO label UNTIL LCE;
        R0=DM(I0,M0), F2=PM(I8,M8);
        R1=R0-R15;
label:  F4=F2+F3;
```

Loop length

Last instruction in loop

Termination condition

Example: FIR filter

- C:

```
for (i=0, f=0; i<N; i++)
  f = f + c[i]*x[i];
```

FIR filter assembler

```
! setup
  I0=_a; I8=_b; ! a[0] (DAG0), b[0] (DAG1)
  M0=1; M8=1 ! Set up increments
! Loop body
  LCNTR=N, DO loopend UNTIL LCE;
  ! Use postincrement mode
  R1=DM(I0,M0), R2=PM(I8,M8);
  R8=R1*R2;
loopend: R12=R12+R8;
```

Optimized FIR filter code

```
  I4=_a; I12=_b;
  R4 = R4 xor R4, R1=DM(I4,M6),
  R2=PM(I12,M14);
  MR0F = R4, MODIFY(I7,M7);
! Start loop
  LCNTR=20, DO(PC,loop) UNTIL LCE;
loop: MR0F=MR0F+42*R1 (SSI), R1=DM(I4,M6),
  R2=PM(I12,M14);
! Loop cleanup
  R0=MR0F;
```

SHARC subroutine calls

- Use CALL instruction:

```
CALL foo;
```

- Can use absolute, indirect, PC-relative addressing modes.
- Return using RTS instruction.

PC stack

PC stack: 30 locations X 24 instructions.

Return addresses for subroutines, interrupt service routines, loops held in PC stack.

Example: C function

- C:

```
void f1(int a) { f2(a); }
```

- Assembler:

```
f1:  R0=DM(I1,-1); ! Load arg into R0
  DM(I1,M1)=R0; ! Push f2's arg
  CALL f2;
  MODIFY(I1,-1); ! Pop element
  RTS;
```

Important programming reminders

- Non-delayed branches (JUMP, CALL, RTS, RTI) do not execute 2 following instructions. Delayed branches are available.
- Cache miss costs at least one cycle to allow program memory bus to complete.

Important programming reminders, cont'd

- Extra cache misses in loops:
  - misses on first and last loop iteration if data memory is accessed in last 2 instructions of loop;
  - 3 misses if loop has only one instruction which requires a program memory bus access.
- 1-instr. loops should be executed 3 times, 2-instr. loops 2 times to avoid NOPs.

Important programming reminders, cont'd

- NOPs added for DAG register write followed by DAG data addressing in same register bank.
- Can program fixed wait states or ACK.
- Interrupt does not occur until 2 instructions after delayed branch.
- Initialize circular buffer by setting L to positive value, loading B to base.

Important programming reminders, cont'd

- Some DAG register transfers are disallowed.
- When given 2 writes to same register file in same cycle, only one actually occurs.
- Fixed- to floating-point conversion always rounds to 40 bits.
- Only DM bus can access all memory spaces.

Important programming reminders, cont'd

- When mixing 32-bit and 48-bit words in a block, all instructions must be below data.
- 16-bit short words are extended to 32 bits.
- For dual data access, use DM for data-only access, PM for mixed data/instruction block. Instruction comes from cache.
- A variety of conditions cause stalls.