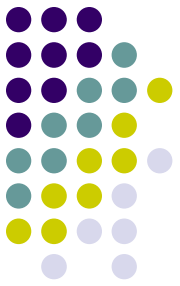# **Processor classes**

- General purpose processors

- Special purpose processors
  - **DSPs**
  - Microcontrollers
  - GPUs...

Example of a typical code in DSP

```
float aryA[N];
float aryB[N];
float aryC[N];

void conv() {
  int i;
  for (i = 0; i < N; ++i) {
    aryC[i] = aryA[i] * aryB[i];
  }
}
```
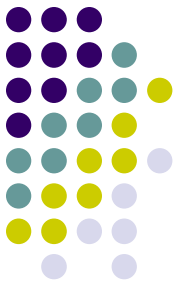
Fixed point arithmetic

```
1.000 (-1)    int a = 0x8; //1000    _Fract a = -1.0r;
0.011 (3/8)   int b = 0x3; //0011    _Fract b = 0.375r;


11.101000     int c = fp_mul(a, b);  _Fract c = a * b;
    << 1
1.101000
```

```
_Fract
long _Fract
_Accum
long _Accum
```

"Embedded C" - Extensions for the
programming language C to support
embedded processors

```
0.25r, 0.5lr, 5.6k, 50.7lk, 1.5r
```

Accumulators

0.5 + 0.75 - 0.3 - 0.8 - 0.6 - 0.4 - 0.25 + 0.5 = -0.6

0.5 + 0.75 = 1.25
0.5 + 0.75 - 0.3 = 0.95
0.5 + 0.75 - 0.3 - 0.8 = 0.15
0.5 + 0.75 - 0.3 - 0.8 - 0.6 = -0.45
0.5 + 0.75 - 0.3 - 0.8 - 0.6 - 0.4 = -0.85
0.5 + 0.75 - 0.3 - 0.8 - 0.6 - 0.4 - 0.25 = -1.1
0.5 + 0.75 - 0.3 - 0.8 - 0.6 - 0.4 - 0.25 + 0.5 = -0.6
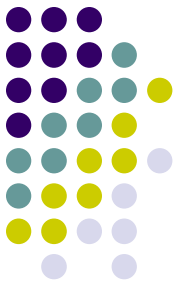
```
float aryA[N];
float aryB[N];
float aryC[N];

void conv() {
  int i;
  for (i = 0; i < N; ++i) {
    aryC[i] = aryA[i] * aryB[i];
  }
}

_Fract aryA[N];
_Fract aryB[N];
_Fract aryC[N];

void conv() {
  int i;
  for (i = 0; i < N; ++i) {
    aryC[i] = aryA[i] * aryB[i];
  }
}
```

```
_conv:
  a1 = 0
start:
  a1 > 30
  if (T) jmp end:
    i0 = _aryA + a1
    x0 = mem[i0]
    i0 = _aryB + a1
    y0 = mem[i0]
    a0 = x0 * y0
    i0 = _aryC + a1
    mem[i0] = a0
    a1 = a1 + 1
    jmp start
end:
    i7 = mem[i6]
    jmp i7
```
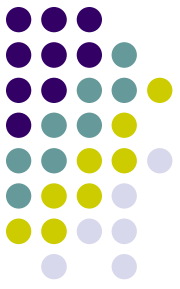
Address generators

```
...ary[0];        p = ary;
...ary[2];        ...*p; p+=2
...ary[1];        ...*p--;
...ary[2];        ...*p++;
...ary[3];        ...*p++;
                  ...*p;
```

```
struct s          struct s* ps;      struct s
{                                    {
  int a;            ...ps->d;          int d;
  int b;            ...ps->a;          int a;
  int c;            ...ps->c;          int c;
  int d;            ...ps->b;          int b;
};                                   };
```

```
_Fract aryA[N];
_Fract aryB[N];
_Fract aryC[N];

void conv() {
  int i;
  for (i = 0; i < N; ++i) {
    aryC[i] = aryA[i] * aryB[i];
  }
}


void conv() {
  int i;
  _Fract* pA = &aryA[0];
  _Fract* pB = &aryB[0];
  _Fract* pC = &aryC[0];
  for (i = 0; i < N; ++i) {
    *pC++ = *pA++ * *pB++;
  }
}
```

```
_conv:
  a1 = 0
start:
  a1 > 30
  if (T) jmp end:
    i0 = _aryA + a1
    x0 = mem[i0]
    i0 = _aryB + a1
    y0 = mem[i0]
    a0 = x0 * y0
    i0 = _aryC + a1
    mem[i0] = a0
    a1 = a1 + 1
    jmp start
end:
    i7 = mem[i6]
    jmp i7
```

```
_conv:
  a1 = 0
start:
  a1 > 30
  if (T) jmp end:
    x0 = mem[i0]
    i0 = i0 + 1
    y0 = mem[i4]
    i4 = i4 + 1
    a0 = x0 * y0
    mem[i1] = a0
    i1 = i1 + 1
    a1 = a1 + 1
    jmp start
end:
    i7 = mem[i6]
    jmp i7
```

```
_Fract aryA[N];
_Fract aryB[N];
_Fract aryC[N];

void conv() {
  int i;
  for (i = 0; i < N; ++i) {
    aryC[i] = aryA[i] * aryB[i];
  }
}

void conv() {
  int i;
  _Fract* pA = &aryA[0];
  _Fract* pB = &aryB[0];
  _Fract* pC = &aryC[0];
  for (i = 0; i < N; ++i) {
    *pC++ = *pA++ * *pB++;
  }
}
```
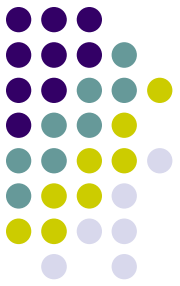
```
_conv:
  a1 = 0
start:
  a1 > 30
  if (T) jmp end:
    x0 = mem[i0]
    i0 = i0 + 1
    y0 = mem[i4]
    i4 = i4 + 1
    a0 = x0 * y0
    mem[i1] = a0
    i1 = i1 + 1
    a1 = a1 + 1
    jmp start
end:
    i7 = mem[i6]
    jmp i7
```
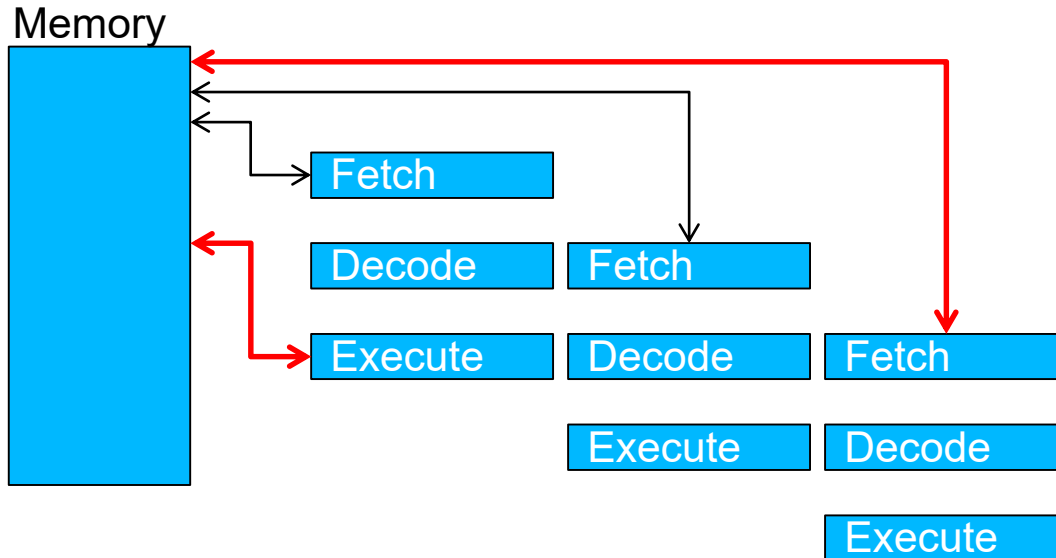
```
_conv:
  a1 = 0
start:
  a1 > 30
  if (T) jmp end:
    x0 = mem[i0]; i0 += 1
    y0 = mem[i4]; i4 += 1
    a0 = x0 * y0
    mem[i1] = a0; i1 += 1
    a1 = a1 + 1
    jmp start
end:
    i7 = mem[i6]
    jmp i7
```
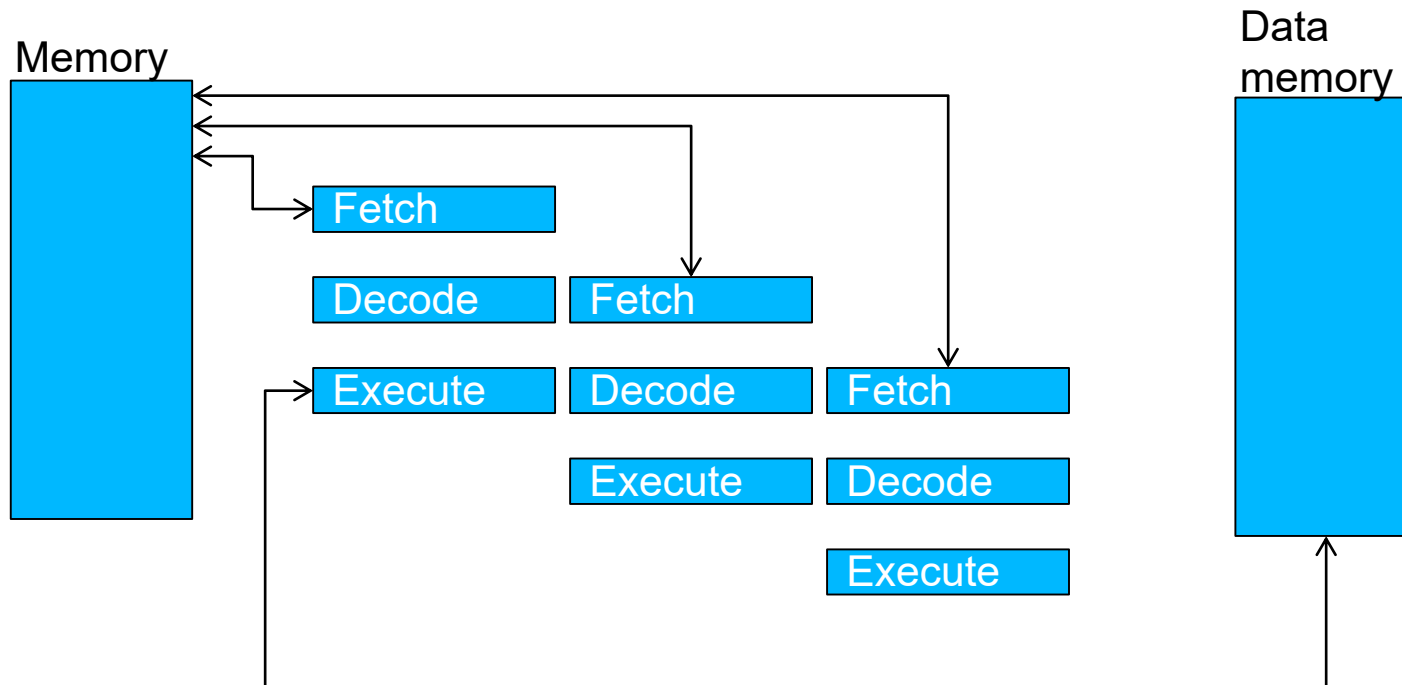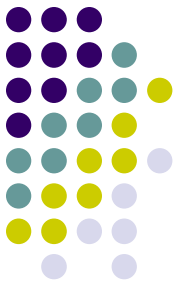
Harvard architecture

Memory

Fetch

Decode

Execute

Harvard architecture

Memory

Fetch

Decode   Fetch

Execute   Decode   Fetch

Execute   Decode

Execute

Harvard architecture

Memory

Data memory
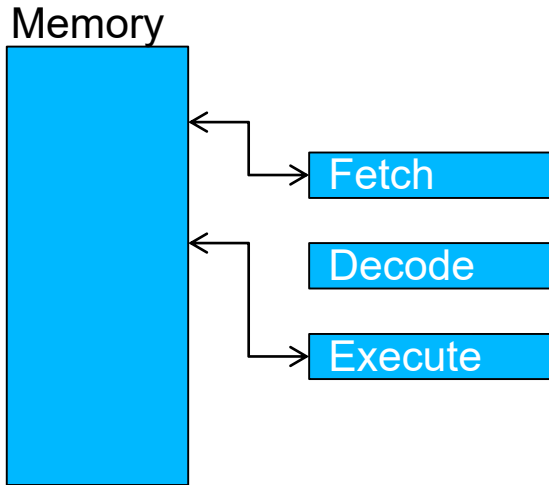
Fetch		
Decode	Fetch	
Execute	Decode	Fetch
	Execute	Decode
		Execute

```
c = a * b
How many memory accesses?
```

Harvard architecture

Memory

Data memory 1

Data memory 2

Fetch

Decode | Fetch

Execute | Decode | Fetch

Execute | Decode

Execute
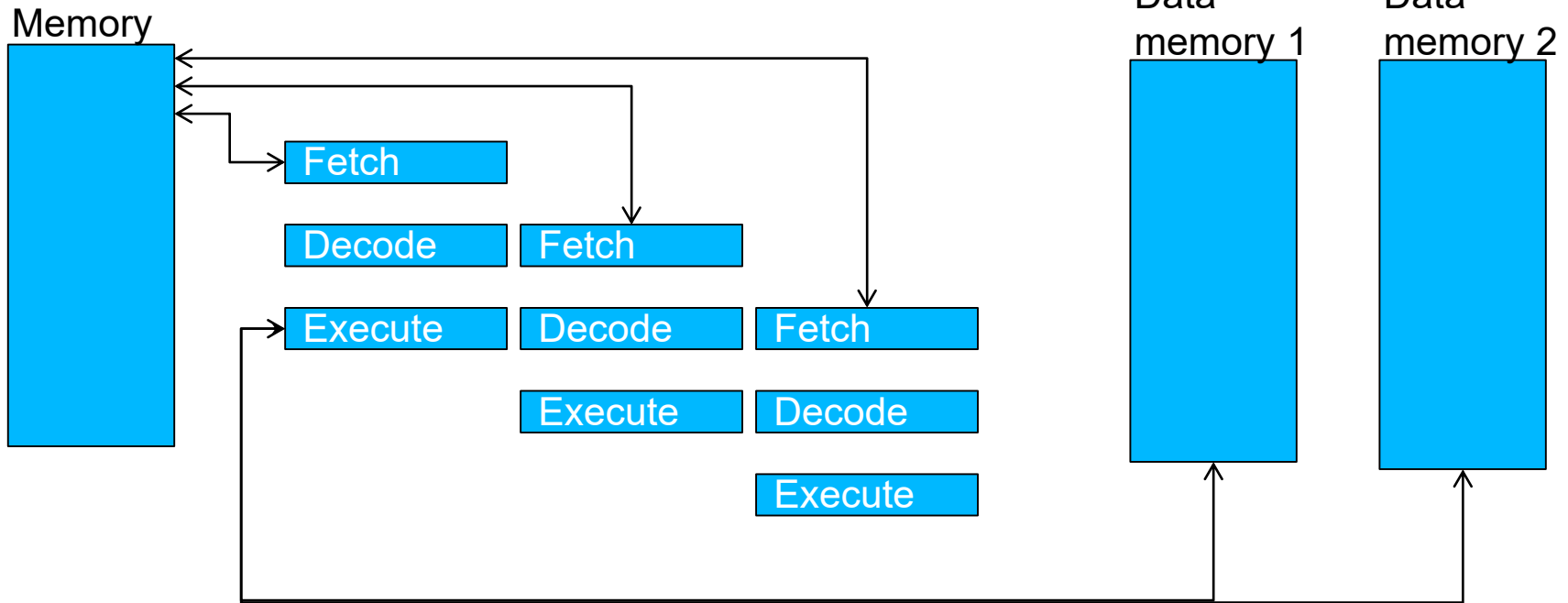
```
__memA int ary1[100];
__memB int ary2[100];
```

```
_Fract aryA[N];
_Fract aryB[N];
_Fract aryC[N];

void conv() {
  int i;
  _Fract* pA = &aryA[0];
  _Fract* pB = &aryB[0];
  _Fract* pC = &aryC[0];
  for (i = 0; i < N; ++i) {
    *pC++ = *pA++ * *pB++;
  }
}
```

```
__memX _Fract aryA[N];
__memY _Fract aryB[N];
__memX _Fract aryC[N];

void conv() {
  int i;
  __memX _Fract* pA = &aryA[0];
  __memY _Fract* pB = &aryB[0];
  __memX _Fract* pC = &aryC[0];
  for (i = 0; i < N; ++i) {
    *pC++ = *pA++ * *pB++;
  }
}
```

```
__memX _Fract aryA[N];
__memY _Fract aryB[N];
__memX _Fract aryC[N];

void conv() {
  int i;
  for (i = 0; i < N; ++i) {
    aryC[i] = aryA[i] * aryB[i];
  }
}
```
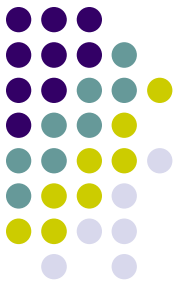
Number of instructions
14 (11 in loop)
11 (8 in loop)
10 (7 in loop)

```
_conv:
  a1 = 0
start:
  a1 > 30
  if (T) jmp end:
    x0 = mem[i0]; i0 += 1
    y0 = mem[i4]; i4 += 1
    a0 = x0 * y0
    mem[i1] = a0; i1 += 1
    a1 = a1 + 1
    jmp start
end:
    i7 = mem[i6]
    jmp i7
```
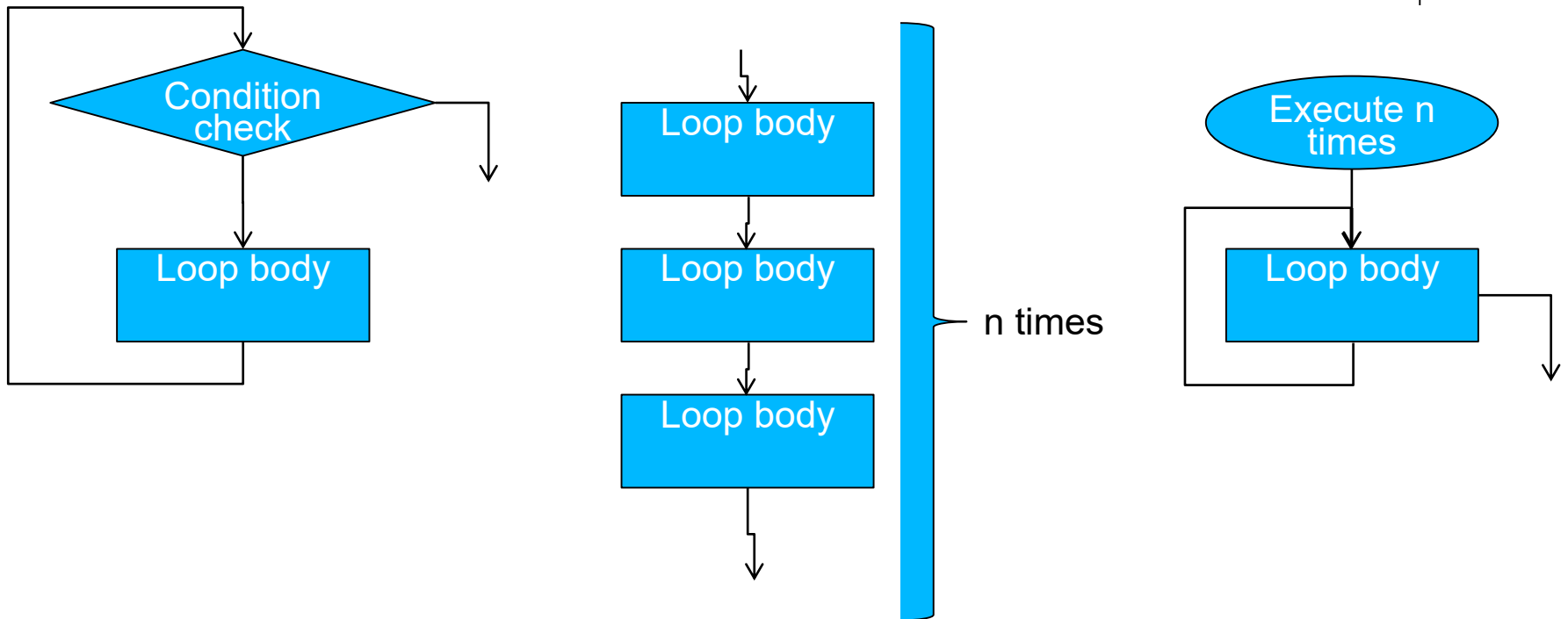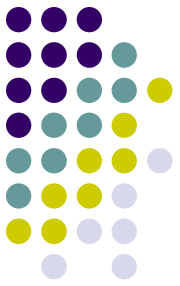
```
_conv:
  a1 = 0
start:
  a1 > 30
  if (T) jmp end:
    x0 = xmem[i0]; i0 += 1; y0 = ymem[i4]; i4 += 1
    a0 = x0 * y0
    xmem[i1] = a0; i1 += 1
    a1 = a1 + 1
    jmp start
end:
    i7 = mem[i6]
    jmp i7
```

Hardware loops

- Loops for which the number of iteration is known in compile time
- Loops for which the number of iteration is known before the first iteration (but can vary between to encounters of the same loop)

Hardware loops

```
for (i = 0; i < NUMBER_OF_ITERATIONS; i++)
for (i = NUMBER_OF_ITERATIONS; i > 0; i--)
for (i = 0; i++ < NUMBER_OF_ITERATIONS; )
for (i = NUMBER_OF_ITERATIONS; i-- > 0; )
for (i = 0; i < NUMBER_OF_ITERATIONS; ++i)
for (i = NUMBER_OF_ITERATIONS;i>0; --i)
for (i = 0; ++i < NUMBER_OF_ITERATIONS + 1; )
for (i = NUMBER_OF_ITERATIONS + 1; --i > 0; )
```

```
__memX _Fract aryA[N];
__memY _Fract aryB[N];
__memX _Fract aryC[N];

void conv() {
  int i;
  for (i = 0; i < N; ++i) {
    aryC[i] = aryA[i] * aryB[i];
  }
}
```

```
__memX _Fract aryA[N];
__memY _Fract aryB[N];
__memX _Fract aryC[N];

void conv() {
  int i;
  __memX _Fract* pA = &aryA[0];
  __memY _Fract* pB = &aryB[0];
  __memX _Fract* pC = &aryC[0];
  for (i = 0; i < N; ++i) {
    *pC++ = *pA++ * *pB++;
  }
}
```
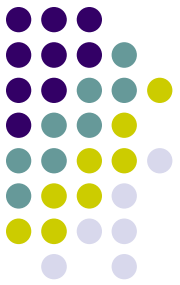
Number of instructions
14 (11 in loop)
11 (8 in loop)
10 (7 in loop)
5 (3 in loop)

```
_conv:
  a1 = 0
start:
  a1 > 30
  if (T) jmp end:
    x0 = xmem[i0]; i0 += 1; y0 = ymem[i4]; i4 += 1
    a0 = x0 * y0
    xmem[i1] = a0; i1 += 1
    a1 = a1 + 1
    jmp start
end:
    i7 = mem[i6]
    jmp i7
```

```
                            _conv:
                              hw_loop(31), end
                                x0 = xmem[i0]; i0 += 1; y0 = ymem[i4]; i4 += 1
                                a0 = x0 * y0
                                xmem[i1] = a0; i1 += 1
                            end:
                                i7 = mem[i6]
                                jmp i7
```

Hardware supported function call stack

```
push pc
jmp addr

pop reg
jmp reg
```

```
call addr

ret
```

Number of instructions
14 (11 in loop)
11 (8 in loop)
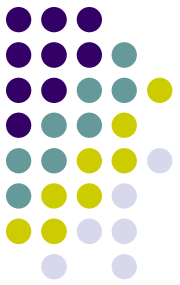10 (7 in loop)
5 (3 in loop)
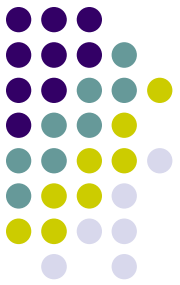4 (3 in loop)

```
_conv:
  hw_loop(31), end
    x0 = xmem[i0]; i0 += 1; y0 = ymem[i4]; i4 += 1
    a0 = x0 * y0
    xmem[i1] = a0; i1 += 1
end:
    i7 = mem[i6]
    jmp i7


_conv:
  hw_loop(31), end
    x0 = xmem[i0]; i0 += 1; y0 = ymem[i4]; i4 += 1
    a0 = x0 * y0
    xmem[i1] = a0; i1 += 1
end:
    ret



i6 += 1
xmem[i6] = pc                        call conv
jmp conv
i6 -= 1
```
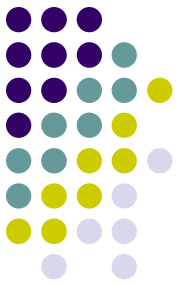
```
_conv:
  hw_loop(31), end
    x0 = xmem[i0]; i0 += 1; y0 = ymem[i4]; i4 += 1
    a0 = x0 * y0
    xmem[i1] = a0; i1 += 1
end:
    ret
```

Number of instructions
14 (11 in loop)

...
4 (3 in loop)
4 (2 in loop)

```
_conv:
  x0 = xmem[i0]; i0 += 1; y0 = ymem[i4]; i4 += 1
  hw_loop(31), end
    x0 = xmem[i0]; i0 += 1; y0 = ymem[i4]; i4 += 1; a0 = x0 * y0
    xmem[i1] = a0; i1 += 1
end:
    ret
```
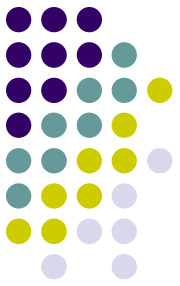
Hardware supported circular buffer

```
__attribute__(align(128))


p = ary;
for (...)
{
    ...*p...
    p = CIRC_INC(p, MOD128, 1);
}
```

Vector operations GCC
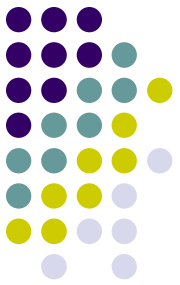
```
int8_t A[200];
int8_t B[200];
int8_t C[200];
for (i = 0; i < 200; ++i)
{
    C[i] = A[i] * B[i];
}


int8_t __attribute__ ((vector_size(8))) A[25];
int8_t __attribute__ ((vector_size(8))) B[25];
int8_t __attribute__ ((vector_size(8))) C[25];
for (i = 0; i < 25; ++i)
{
    C[i] = A[i] * B[i];
}


int8_t __attribute__ ((vector_size(8))) x = {1, 2, 3, 4, 5, 6, 7, 8};
```

Vector operations GCC – ARM NEON

```
int8_t A[200];
int8_t B[200];
int8_t C[200];
for (i = 0; i < 200; ++i)
{
    C[i] = A[i] + B[i];
}


int8_t A[200];
int8_t B[200];
int8_t C[200];
for (i = 0; i < 25; i += 8)
{
    _Int8x8 a = vld1_s8(&A[i]); // int8x8_t, through typdef in header
    _Int8x8 b = vld1_s8(&B[i]);
    vst1_s8(&C[i], vadd_s8(a, b)); // although a + b is also OK here
}
```