Data Representation

for

Assembly Programmers

ကွန်ပြူတာတွေက data တွေကို binary bits အဖြစ် အစဉ်လိုက် သိမ်းဆည်းပါတယ်။ Bit ဆိုတာ ကွန်ပြူတာသိမ်းဆည်းနိုင်တဲ့ အသေးငယ်ဆုံးပမာဏ ဖြစ်ပါတယ်။ Bit ၁ ခု မှာ 1 (သို့) 0 ပဲ ဆန့်ပါတယ်။ Bits ၈ ခုစုထားတဲ့အစုကို byte လို့ ခေါ်ပါတယ်။ ထိုနည်းတူစွာ 16 bits ကို halfword၊ 32 bits ကို word၊ 64 bits ကို double-word လို့ ခေါ်ပါတယ်။ Cortex-M မှာပါတဲ့ register တွေက 32-bit register ဖြစ်တဲ့အတွက်ကြောင့် အများဆုံး 32-bits အထိ သိမ်းနိုင်ပါတယ်။ C language မှာပါတဲ့ integer data type နဲ့ ညီပါတယ်။

Integer data type တွေက Unsigned integers နဲ့ Signed integers ဆိုပြီး နှစ်မျိုးရှိပါတယ်။ Unsigned integer ဆိုတာ အနုတ်လက္ခဏာ(minus) မပါတဲ့ကိန်းတွေဖြစ်ပြီး n-bit မှာ range $[0, 2^n - 1]$ ရှိပါတယ်။ Signed integer ဆိုတာ အနုတ်ရော၊ အပေါင်းရော ပါတဲ့ကိန်းတွေဖြစ်ပြီး n-bit မှာ range $[-2^{n-1}, 2^{n-1} - 1]$ ရှိပါတယ်။

Unsigned number တွေက အနုတ်ကိန်းမပါတဲ့အတွက်ကြောင့် ကွန်ပြူတာမှာ သိမ်းဆည်းရတာ ထွေထွေထူးထူး မရှိပါဘူး။ ရိုးရှင်းပါတယ်။ ဥပမာ 1 ကို သိမ်းချင်ရင် 0x00000001 ဆိုပြီး တိုက်ရိုက်သိမ်းပါတယ်။

Unsigned number တွေကျတော့ အနုတ်ကိန်းပါလာပြီဖြစ်တဲ့အတွက် unsigned number တွေလို သိမ်းလို့မရတော့ပါဘူး။ ဘာလို့လဲဆိုတော့ ကွန်ပြူတာ cpu က အနုတ်တွေ၊ အပေါင်းတွေ နားမလည်ပါဘူး။ သူနားလည်တာက binary bits (0, 1) တွေပဲ ဖြစ်ပါတယ်။ အဲဒါကြောင့် sign-and-magnitude၊ one'scomplement၊ two's complement တစ်ခုခုသုံးပြီး သိမ်းရပါတယ်။ လက်ရှိကွန်ပြူတာတွေသုံးနေတာကတော့ Two's Complement ပဲ ဖြစ်ပါတယ်။

Two's complement မှာ အပေါင်းကိန်းတွေကို သူ့ပကတိတန်ဖိုးအတိုင်း ထားပါတယ်။ အနုတ်ကိန်းတွေကိုတော့ အနည်းငယ် ပြင်ပါတယ်။

- အနုတ်ကိန်းရဲ့ အပေါင်းတန်ဖိုးရဲ့ binary form ကို ပြောင်းပြန်လှန်ပစ်ပါတယ်။
- ပြီးလျှင် 1 နှင့် ပြန်ပေါင်းပါတယ်။

ဥပမာ 4-bit system တစ်ခုမှာ -3 ကို memory မှာသွားသိမ်းရင် အပေါ်က step တွေကိုအသုံးပြုပြီး two's complement အနေနဲ့ သိမ်းမှာဖြစ်ပါတယ်။ -3 ရဲ့ အပေါင်းတန်ဖိုး 3 (binary form – 0011) ကို ပြောင်းပြန်လှန်လို့ရတဲ့ 1100 ကို 1 ပေါင်းလိုက်တော့ 1101 ရပါတယ်။ အဲဒါကြောင့် -3 ဆိုတာနဲ့ memory မှာ 1101 သိမ်းမှာဖြစ်ပါတယ်။

Binary Bit String	Two's Complement
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

Binary Bit String	Two's Complement
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

4-bit system, signed number မှာ range က $[-2^{4-1}, 2^{4-1} - 1] = [-8, 7]$ ထိ ရှိပါတယ်။

Two's complement ကို အသုံးပြုခြင်းအားဖြင့် Hardware အတွက် သက်သာစေပါတယ်။ ဘာလို့လဲဆိုတော့ input operands တွေက signed ပဲဖြစ်ဖြစ် unsigned ပဲဖြစ်ဖြစ် ပေါင်းခြင်း၊ နုတ်ခြင်း၊ မြှောက်ခြင်းတွေကို အမှားအယွင်းမရှိဘဲ လုပ်ထုံးတူလုပ်နိုင်လို့ပဲ ဖြစ်ပါတယ်။

Unsigned number တွေကို ပေါင်း (သို့) နုတ်လျှင် result ပေါ်မူတည်ပြီး processor က carry flag ကို update ပါတယ်။

- Unsigned number တွေကိုပေါင်းတဲ့အခါ result က maximum range ထက်ကျော်လျှင် carry flag က 1 ထဖြစ်ပါတယ်။
- Unsigned number တွေကိုနုတ်တဲ့အခါ result က အနုတ် (သို့) minimum range အောက်ငယ်လျှင် borrow flag က 1 ထဖြစ်ပါတယ်။
- Carry နဲ့ borrow က hardware တစ်ခုတည်း ဖြစ်ပါတယ်။ Carry = 1 ဖြစ်ရင် borrow = 0၊ carry = 0 ဆိုရင် borrow = 1။

For unsigned subtraction,
Carry = NOT Borrow

- Unsigned addition မှာ carry က 1 ဆိုရင် ထွက်လာတဲ့ result က မှားပါတယ်။
- Unsigned subtraction မှာ carry က 1 ဆိုရင် ထွက်လာတဲ့ result က မှန်ပါတယ်။

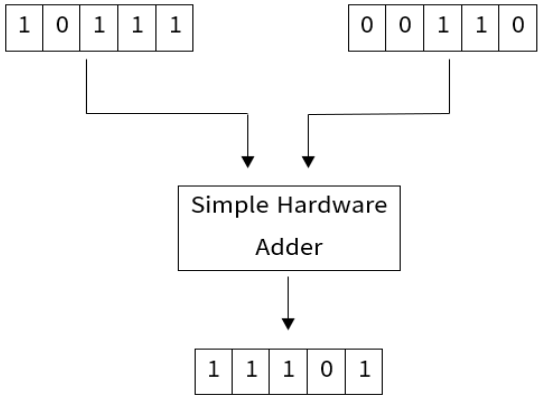
Signed number တွေကို ပေါင်း (သို့) နုတ်လျှင် result ပေါ်မူတည်ပြီး processor က overflow flag ကို update ပါတယ်။

- Result က range $[-2^{n-1}, 2^{n-1}-1]$ ရဲ့အပြင်ရောက်လျှင် overflow ဖြစ်ပါတယ်။
- Signed number တွေကိုပေါင်းတဲ့အခါ…
  - အပေါင်းကိန်းနှစ်ခု ပေါင်းခြင်းရဲ့ရလာဒ်က အပေါင်းကိန်းမဖြစ်လျှင် (သို့)
  - အနုတ်ကိန်းနှစ်ခု ပေါင်းခြင်းရဲ့ရလာဒ်က အနုတ်ကိန်းမဖြစ်လျှင်
  overflow ထဖြစ်ပါတယ်။
- Signed number တွေကိုနုတ်တဲ့အခါ…
  - အနုတ်ကိန်းထဲကနေ အပေါင်းကိန်းကို နုတ်ခြင်းရဲ့ရလာဒ်က အပေါင်းကိန်း (သို့)
  - အပေါင်းကိန်းထဲကနေ အနုတ်ကိန်းကို နုတ်ခြင်းရဲ့ရလာဒ်က အနုတ်ကိန်း ဖြစ်နေလျှင်
  overflow ထဖြစ်ပါတယ်။
- လက္ခဏာမတူတဲ့ဂဏန်းနှစ်ခုကို ပေါင်းလျှင် (သို့) လက္ခဏာတူတဲ့ဂဏန်းနှစ်ခုကို နုတ်လျှင် overflow မဖြစ်ပါဘူး။
- Unsigned addition or subtraction မှာ overflow က 1 ဆိုရင် result က မှားပါတယ်။

Two's complement သုံးခြင်းက hardware ရဲ့အလုပ်ကိုလွယ်ကူစေပါတယ်။ ပေါင်း၊ နုတ်၊ မြှောက်၊ စား တွေကို hardware က ဘယ်လိုအလုပ်လုပ်လဲဆိုတာ ကြည့်ရအောင်။ ။

Two's complement addition

0b10111 နဲ့ 0b00110 တို့ကို 5-bit system တစ်ခုမှာ ပေါင်းမည်ဆိုကြပါစို့။ Hardware adder က ဝင်လာသမျှကို unsigned လိုသတ်မှတ်ပြီး အလုပ်လုပ်ပါတယ်။

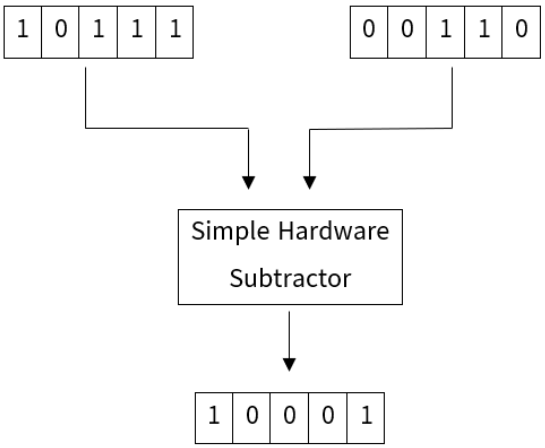


0b11101 ဟာ unsigned number ဆိုရင် 29 ဖြစ်ပြီးတော့ signed number ဆိုရင် -3 ဖြစ်ပါတယ်။ ဒါပေမယ့် two's complement ကို သုံးထားတဲ့အတွက်ကြောင့် နှစ်ခုစလုံးအတွက် အဖြေမှန်ပါတယ်။

Simple Addition (ignore the sign)	Unsigned Addition	Signed Addition
1 0 1 1 1	23	-9
+ 0 0 1 1 0	+ 6	+ 6
1 1 1 0 1	29	-3

Two's complement subtraction

Hardware subtractor က ဝင်လာသမျှကို unsigned လိုသတ်မှတ်ပြီး အလုပ်လုပ်ပါတယ်။



0b10001 ဟာ unsigned number ဆိုရင် 17 ဖြစ်ပြီးတော့ signed number ဆိုရင် -15 ဖြစ်ပါတယ်။ ဒါပေမယ့် two's complement ကို သုံးထားတဲ့အတွက်ကြောင့် နှစ်ခုစလုံးအတွက် အဖြေမှန်ပါတယ်။

Simple Subtraction (ignore the sign)	Unsigned Subtraction	Signed Subtraction
1 0 1 1 1	23	-9
- 0 0 1 1 0	- 6	- 6
1 0 0 0 1	17	-15

Two's complement multiplication

တကယ်လို့ output ရဲ့ bit အရေအတွက်ကို input ရဲ့ bit အရေအတွက်အတိုင်းပဲ ယူမယ်ဆိုရင် signed multiplication နဲ့ unsigned multiplication တို့ဟာ တူညီတဲ့ hardware တစ်ခုကိုပဲ သုံးပါတယ်။

input ကို signed အနေနဲ့သုံးမယ်ဆိုရင် 0b00011 = 3, 0b11101 = -3 ဖြစ်ပြီးတော့ product 0b10111 = -9 ဆိုပြီး အဖြေမှန်ထွက်ပါတယ်။ တစ်ဖက်မှာတော့ unsigned အနေနဲ့သုံးမယ်ဆိုရင်တော့ 0b000111 = 3, 0b11101 = 29 ဖြစ်ပြီးတော့ product 0b10111 = 23 ဆိုပြီး ထွက်ပါတယ်။ ဒါပေမဲ့ အဲဒါဟာ မှားတယ်လို့ မဆိုလိုပါဘူး။ ဘာလို့လဲဆိုတော့ output ရဲ့ bit အရေအတွက်ကို input ရဲ့ bit အရေအတွက်အတိုင်းပဲ ယူထားလို့ input bit ထက်ကျော်တဲ့ bit တွေကို ဖယ်ရှားထားလို့သာ ဖြစ်ပါတယ်။ (ကြီးမားတဲ့ကိန်းတွေဆိုရင် UMULL SMULL တို့သုံးပြီး register နှစ်ခုမှာ သိမ်းရပါတယ်။)

```
          0 0 0 1 1        multiplicand
      x   1 1 1 0 1    x   multiplier
    -----------------   ---------------
          0 0 0 1 1
        0 0 0 0 0
      0 0 0 1 1
    0 0 0 1 1
  0 0 0 1 1
-----------------------   ---------------
0 0 1 0 1 0 1 1 1 1          product
```

Two’s complement division

စားခြင်းမှာတော့ ပေါင်း၊ နုတ်၊ မြှောက် တွေလို signed နဲ့ unsigned number တွေဟာ တူညီတဲ့ hardware တစ်ခုကို တိုက်ရိုက်သုံးလို့ မရပါဘူး။ ဥပမာ unsigned number 22 (0b10110) ကို 2 (0b00010) နဲ့စားမယ်ဆိုရင် စားလာဒ် 11 (0b0101) ဆိုပြီး အဖြေမှန်ထွက်ပါတယ်။

```
          0 1 0 1 1        quotient
      ---------------
1 0 | 1 0 1 1 0          dividend
      1 0
      -----
        0 1
        0 0
        -----
          1 1
          1 0
          -----
            1 0
            1 0
            -----
              0          remainder
```

ဒါပေမယ့် signed number -10 (0b10110 in two’s complement) ကို 2 (0b00010 in two’ complement) နဲ့စားမယ်ဆိုရင် စားလာဒ် 11 (0b01011 in two’s complement) ဆိုပြီး အဖြေမှား

ထွက်လာပါတယ်။ အဲ့ကိစ္စဖြေရှင်းဖို့ signed number တွေကို သူ့ရဲ့ အပေါင်းတန်ဖိုး (-10 ဆိုရင် +10) ပြောင်းပြီး unsigned number လိုပဲစားပါတယ်။ ပြီးမှဘဲ စားလာဒ်ကို signed number ပြန်ပြောင်းပါတယ်။

ခြုံငုံသုံးသပ်ရလျှင် two's complement က hardware ကိုလွယ်ကူသက်သာစေပါတယ်။ signed နဲ့ unsigned number တွေဟာ ပေါင်းခြင်း၊ နုတ်ခြင်း၊ မြှောက်ခြင်း တွေကို hardware တစ်ခုကိုပဲအတူတူသုံးလို့ ရသွားပါတယ်။ ဂဏန်းနှစ်ခုကို ပေါင်း (သို့) နုတ်လျှင် signed လား unsigned လား ဆိုတာ hardware က မသိပါဘူး။ Hardware က signed လိုမှတ်ပြီး carry flag ကို update လုပ်ပါတယ်။ တစ်ချိန်တည်းမှာပဲ unsigned လိုမှတ်ပြီး overflow flag ကို update လုပ်ပါတယ်။

Carry flag ကိုသုံးမလား၊ Overflow flag ကိုသုံးမလားဆိုတာ programmer တွေရဲ့ တာဝန်ဖြစ်ပါတယ်။ Unsigned integer အနေနဲ့သုံးချင်ရင် carry flag ကို သုံးရပါမယ်။ Signed integer အနေနဲ့သုံးချင်ရင် overflow flag ကို သုံးရပါမယ်။ (C language ကိုသုံးရင်တော့ ကိုယ် သတ်မှတ်တဲ့ data type ("int" or "unsigned int") ပေါ်မူတည်ပြီး compiler က automatic ရွေးပေးပါတယ်။)

output ရဲ့ bit အရေအတွက်ကို input ရဲ့ bit အရေအတွက်အတိုင်းပဲ လိုချင်တယ်ဆိုရင် signed multiplication နဲ့ unsigned multiplication တို့ဟာ hardware တစ်ခုကိုပဲ မျှသုံးပါတယ်။ စားခြင်းမှာတော့ unsigned division နဲ့ signed division တို့ဟာ hardward တစ်ခုကို တိုက်ရိုက်မျှသုံးလို့မရပါဘူး။

အမှားပါရင် ဝေဖန်ထောက်ပြပေးနိုင်ပါတယ်။ ကျေးဇူးတင်ပါတယ်ခင်ဗျာ။ ။