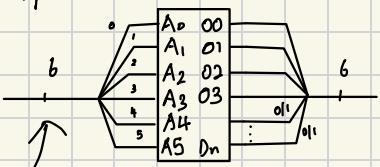
2^6 different things can be represent with 6 bit

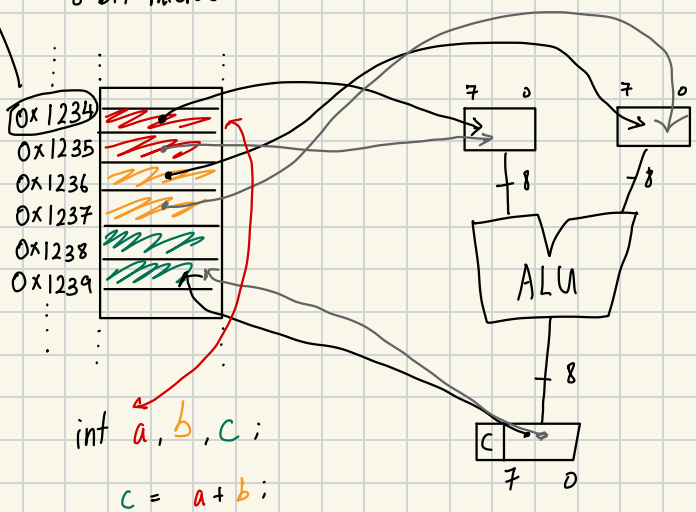


1 byte = 8bit

int 4 byte = 32bit

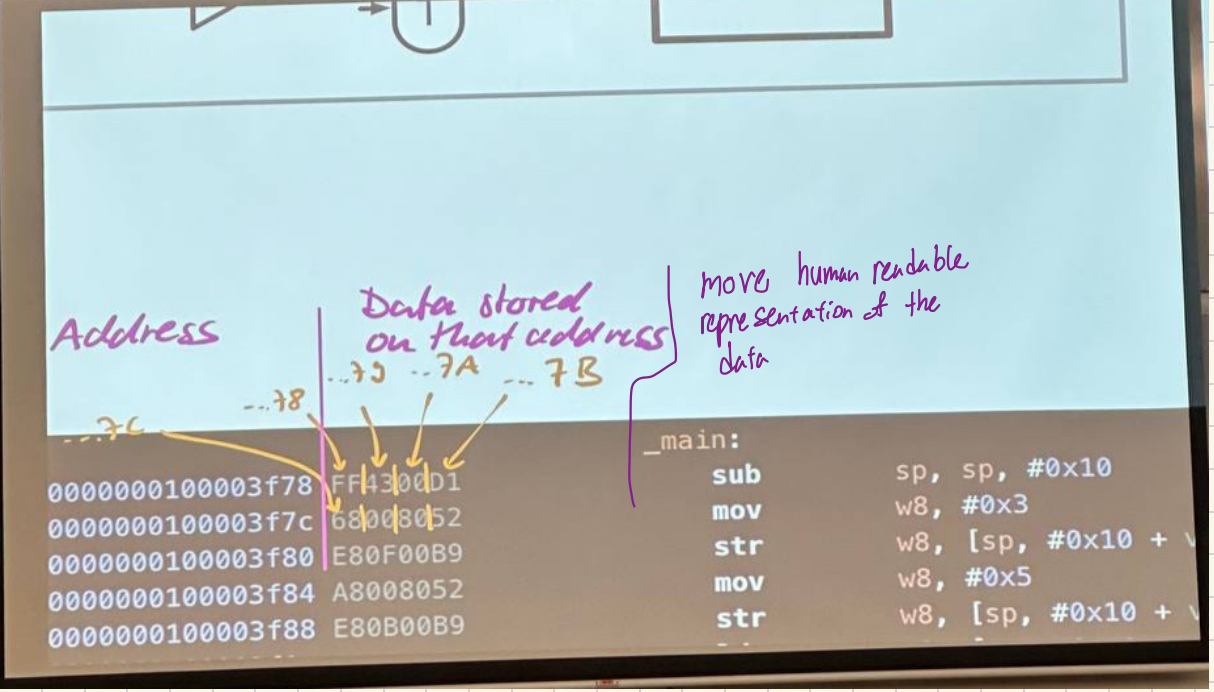
but not always ; depends on compiler

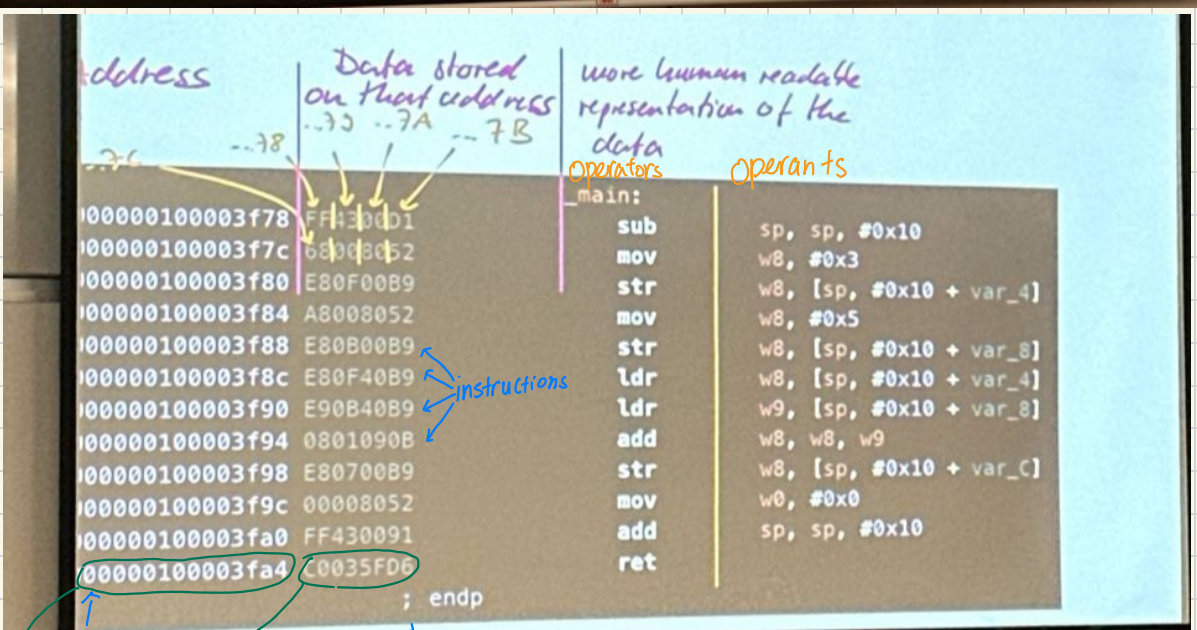
8-bit microcontroller




```
int a, b, c;
c = a + b;
```

```
00   23 69 6E 63 6C 75 64 65 20 3C 73 74 64 69 6F 2E   #include <stdio.
10   68 3E 0A 0A 69 6E 74 20 6D 61 69 6E 28 29 0A 7B   h>..int main().{
20   0A 09 70 72 69 6E 74 66 28 22 48 65 6C 6C 6F 2E   ..printf("Hello.
30   5C 6E 22 29 3B 0A 09 72 65 74 75 72 6E 20 30 3B   \n");..return 0;
40   0A 7D 0A 0A                                       .}..
```





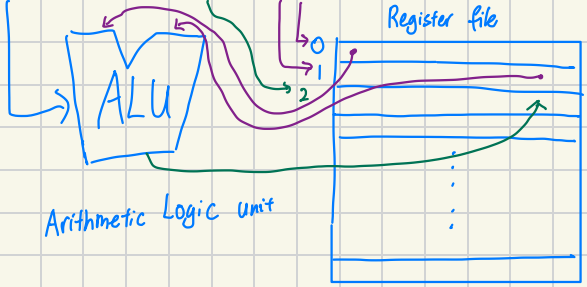
PC
Program Counter

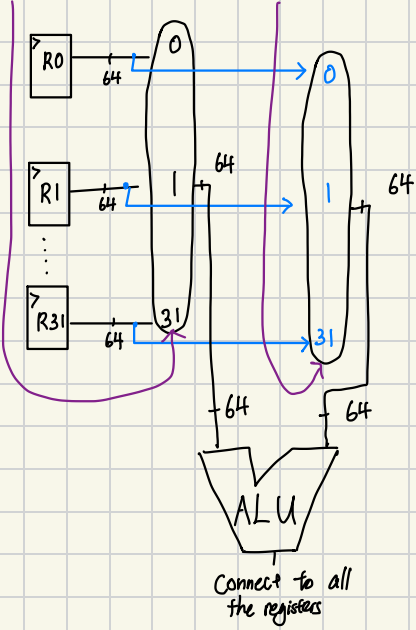
0xC0035FD6

instruction word

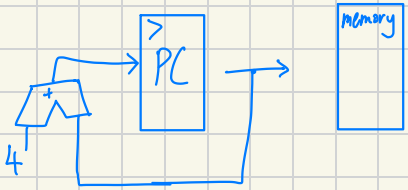
OP.	Destination	Source 1	source 2

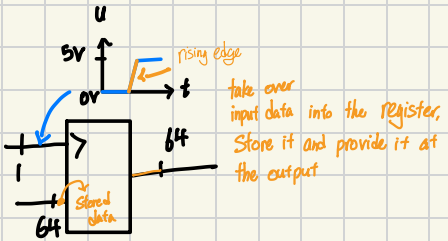
What to do eg. 0x20 : addition
0x28 : Subtraction
⋮ ⋮





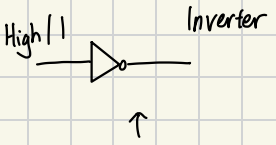
How to adjust program Counter





provide 1, input 1 will show

take multiplexors to select one of out 32



Cmos

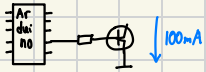
Inverter

output

NPN

Bipolar Logic

$i_C = f(i_B)$



Hex	Binary
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

0010 0011

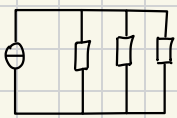
C-File opened in a hex editor:

```
00  23 69 6E 63 6C 75 64 65 20 3C 73 74 64 69 6F 2E   #include <stdio.
10  68 3E 0A 0A 69 6E 74 20 6D 61 69 6E 28 29 0A 7B   h>..int main().{
20  0A 09 70 72 69 6E 74 66 28 22 48 65 6C 6C 6F 2E   ..printf("Hello.
30  5C 6E 22 29 3B 0A 09 72 65 74 75 72 6E 20 30 3B   \n");..return 0;
40  0A 7D 0A 0A                                       .}..
```

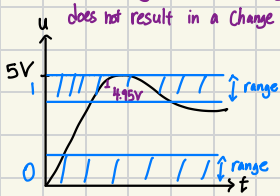
Binary symbols: {0,1}

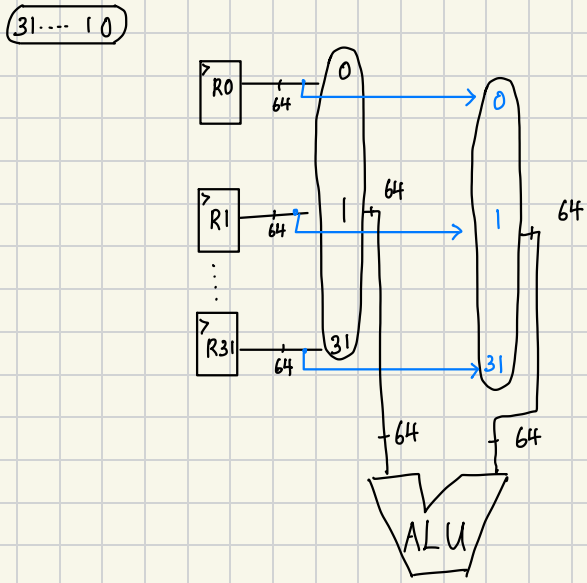
Dec: {0,1,2,3,4,5,6,7,8,9}

Hex: {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}



Range → Single Value ⇒ Voltage drop does not result in a Change of Value




31 ···· 10
R0
R1
R31
64
0
1
31
64
0
1
31
64
64
64
ALU

Bubblesort

nested for-loop

```
for (i = 0; i<n; i++)
{
    for (j = n-1 ; j>i ; j--)
    {
        if (a[j] < a[j-1])
        {
            buf = a[j-1];
            a[j-1] = a[j];
            a[j] = buf;
        }
    }
}
```

Selection Sort

```
for (i=0; i<n; i++)
{
  min_j = i;
  for(j=i; j<n; j++)
  {
    if (a[j] < a[min_j]) min_j = j;
  }
  buf = a[i];
  a[i] = a[min_j];
  a[min_j] = buf;
}
```

Quicksort

```
void quicksort(int a[], int left, int right)
{
  int l, r, p, buf;
  l = left;
  r = right-1;
  p = right;
  do
  {
    while (a[l] < a[p]) l++;
    while (a[r] >= a[p] && l<r) r--;
    if (l<r)
    {
      buf = a[l];
      a[l] = a[r];
      a[r] = buf;
    }
    if (l==r)
    {
      buf = a[r];
      a[r] = a[p];
      a[p] = buf;
      p = l
    }
  } while (l<r);

  if (left < p-1)
    quicksort (a, left, p-1);

  if (p+1 < right)
    quicksort (a, p+1, right);
```

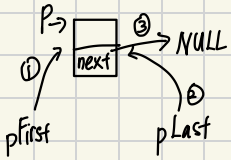
insertion sort

```
for(i = 1; i<n; i++)
{
    for (j=i-1; j>=0; j--)
    {
        if (a[j] > a[j+1])
        {
            buf = a[j];
            a[j] = a[j+1];
            a[j+1] = buf;
        }
    else break;
    }
}
```

Single linked list

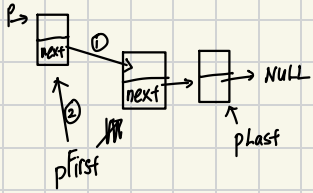
adding to empty list

```
if( pFirst == NULL )
{ pFirst = p;      ①
  pLast = p;       ②
  p→next = NULL;   ③
}
```



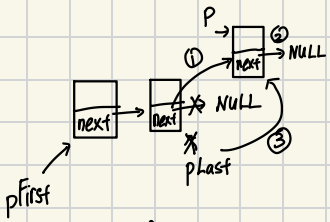
adding beginning of non-empty

```
p→next = pFirst;  ①
pFirst = p;       ②
```



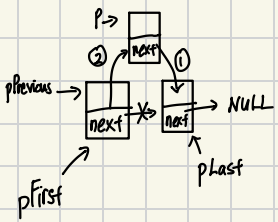
adding at the end of non-empty

```
pLast→next = p;   ①
p→next = NULL;    ②
pLast = p;        ③
```



adding in the middle

```
p→next = pPrevious→next;
pPrevious→next = p;
```



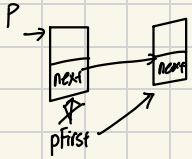
removing, verify with pFirst == pLast

```
p = pFirst;
pFirst = pLast = NULL;
return p;
```

pFirst → NULL pLast → NULL

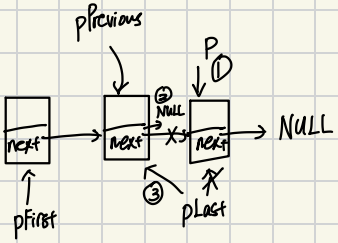
removing first

```
p = pFirst ;
pFirst = pFirst -> next ;
return p ;
```



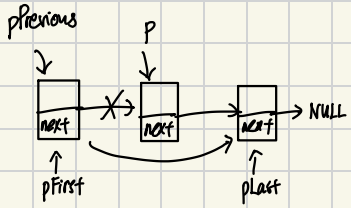
removing last

```
p = pPrevious -> next ; (1)
pPrevious -> next = NULL ; (2)
pLast = pPrevious ; (3)
return p ;
```



remove in between

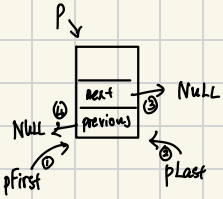
```
p = pPrevious -> next ;
pPrevious -> next = pPrevious -> next -> next ;
return p ;
```



Doubly linked list

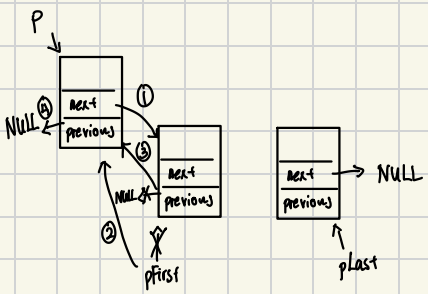
adding to empty list

```
if (pFirst == NULL)
{
    pFirst = p ;        (1)
    pLast = p ;         (2)
    p -> next = NULL ;  (3)
    p -> previous = NULL ; (4)
}
```



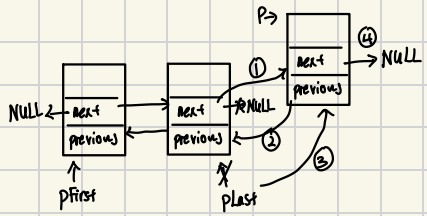
add to beginning

```
p -> next = pFirst ;          (1)
pFirst = p ;                  (2)
p -> next -> previous = p ;   (3)
p -> previous = NULL ;        (4)
```



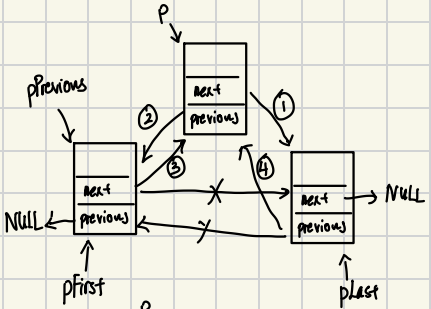
add to end

```
pLast -> next = p ;      ①
p -> previous = pLast ;  ②
pLast = p ;              ③
p -> next = NULL ;       ④
```



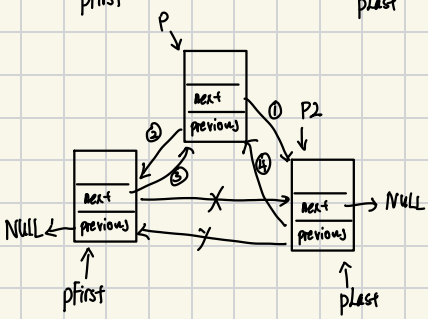
add in middle after pPrevious

```
p -> next = pPrevious -> next ;  ①
p -> previous = pPrevious ;      ②
pPrevious -> next = p ;          ③
p -> next -> previous = p ;      ④
```



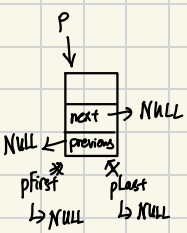
add in middle before p2

```
p -> next = p2                  ①
p -> previous = p2 -> previous  ②
p -> previous -> next = p       ③
p2 -> previous = p              ④
```



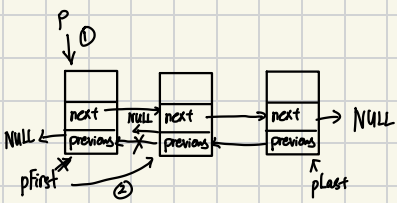
remove only one

```
p = pFirst ;
pFirst = pLast = NULL ;
return p ;
```



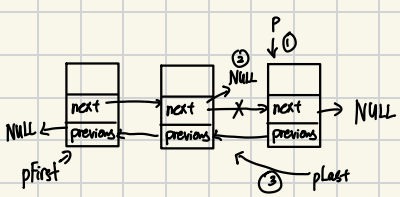
remove first

```
p = pFirst ;                ①
pFirst = pFirst -> next ;   ②
pFirst -> previous = NULL ; ③
return p ;
```



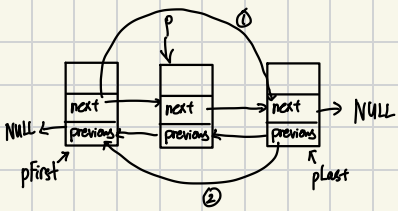
remove last

```
p = pLast ;                          ①
pLast -> previous -> next = NULL ;   ②
pLast = pLast -> previous            ③
return p ;
```



remove in between

```
p-> previous -> next = p-> next ;  ①
p-> next-> previous = p-> previous ;  ②
return p ;
```



linear search

```
int linearsearch (int *a , int length, int number)
{
    for (int i=0; i<length; i++)
        if(a[i] == number) return i;

    return -1;
}
```

binary search

```
int binary_search (int *a , int length, int number)
{ int l, r, m;
  l=0;
  r= length - 1;
  while (l <=r)
  {
      m= (l+r) / 2;
      if (number == a[m]) return m;
      if (a[m] < number) l = m+1
      else r =m-1;
  }
  return -1;
}
```

Searching in Linked list

To store an integer in a doubly linked list

```
struct Integer
{
    int i;
    struct Integer * pNext;
    struct Integer * pPrevious;
};
```

declaration of pointer for first and Last

```
struct Integer *pFirst, *pLast;
```

if list is empty

```
if (pFirst == NULL)
{   pFirst = pLast = p;
    p->pNext = p->pPrevious = NULL;
}
```

if list no empty

```
if (pRun == pFirst && p->i < pRun->i)
```

if add before first

```
if (pRun == pfirst && p->i < pRun->i)
{   p->pNext = pRun;
    pRun->pPrevious = p;
    p->pPrevious = NULL;
    pFirst = p;
    return;
}
```

if add at the end

```
if (pRun->pNext == NULL)
{   pRun->pNext = p;
    p->pPrevious = pRun;
    p->pNext = NULL;
    pLast = p;
    return
}
```

add in middle

```
if (p->i < pRun->pNext->i)
{ p->pNext = pRun->pNext;
  p->pPrevious = pRun;
  p->pNext->pPrevious = p;
  pRun->pNext = p;
  return;
}
```