دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

پروژه امتیازی درس ساختار و زبان کامپیوتر

پیاده سازی توابع سورت با Inline Assembly

اعضای گروه

فاطیما تیمارچی، سامیار لاجوردی، ماهان معصوم زاده، شقایق میرجلیلی

پاییز ۱۴۰۳


بسم الله الرحمن الرحیم

فهرست مطالب

فصل ۱

صورت پروژه

در این پروژه قصد داریم تا توابع سورت Gnome و Bubble را به صورت عادی و Inline با کمک زبان C پیاده سازی کنیم.

۱-۱ گام های پروژه

گام اول: این دو تابع را با کمک زبان C پیاده سازی کنید که ورودی اول تابع تعداد عناصر آرایه و آرگومان دوم آدرس آرایه می باشد.

گام دوم: حال این دو تابع را به صورت Inline Assembly پیاده سازی کنید. آرگومان های ورودی ان مشابه قبل است.

گام سوم: زمان اجرای برنامه های خود را با یکدیگر مقایسه کنید.

۱-۲ اهداف پروژه

۱- آشنایی با نحوه Inline Assembly کد زدن

۲- بهبود سرعت کد با استفاده از این تکنیک

فصل ۲

مراحل انجام پروژه

در این پروژه به بررسی تفاوت زمان اجرا بین توابع پیاده سازی شده توسط زبان c و inline assembly میپردازیم.

gnome sort ۱-۲

روند کلی ۱-۱-۲

در این پروژه ابتدا ورودی توابع را در یک فایل نوشته و سپس تابع را به زبان c مینویسیم و آرابه سورت شده و زمان اجرای کد را در یک فایل متنی مینویسیم .در مرحله بعدی این تابع را به صورت inline assembly مینویسیم و آرایه سورت شده و زمان اجرای کد را در یک فایل متنی مینویسیم . سپس نمودار های تحلیلی زمان اجرایی دو برنامه را کشیده و در انتها تحلیل میکنیم .

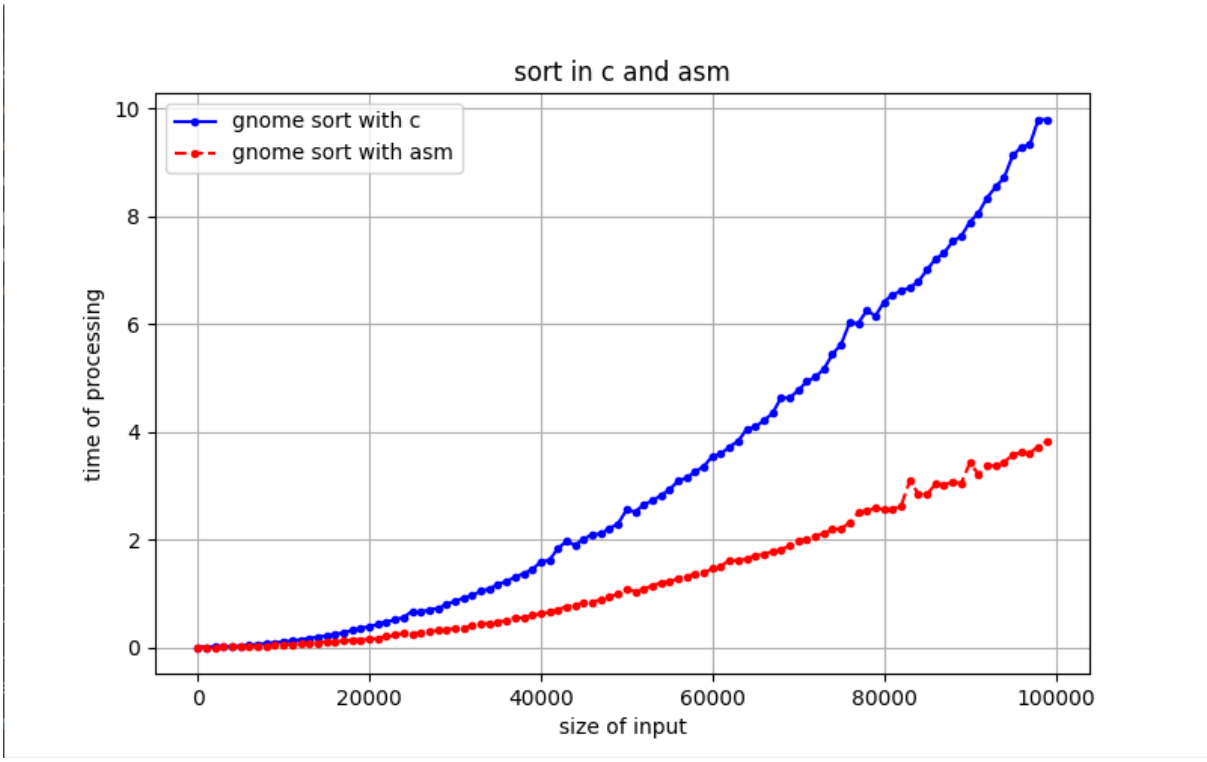
۲-۱-۲ تابع gnome Sort به زبان C

```c
long sort(int n, int *arr)
{
    struct timeval start, end;
    gettimeofday(&start, NULL);

    int index = 0;

    while (index < n)
    {
        if (index == 0)
            index++;
        if (arr[index] >= arr[index - 1])
            index++;
        else
        {
            int temp = 0;
            temp = arr[index];
            arr[index] = arr[index - 1];
            arr[index - 1] = temp;
            index = 1;
        }
    }
    gettimeofday(&end, NULL);
    return (end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec
        - start.tv_usec);
}
```

۲-۱-۳ تابع gnome sort با به کارگیری inline assembly

```
long sort(int n, int *arr)
{
    struct timeval start, end;
    gettimeofday(&start, NULL);
    asm volatile(
        "movl␣$1,␣%%ecx\n\t"
        "start_loop:\n\t"
        "cmpl%%edi,%%ecx\n\t"
        "jge␣end_sort\n\t"
        "movl␣(%%esi,␣%%ecx,␣4),␣%%eax\n\t"
        "movl␣-4(%%esi,␣%%ecx,␣4),␣%%ebx\n\t"
        "cmpl␣%%eax,␣%%ebx\n\t"
        "jle␣no_swap\n\t"

        "movl␣%%eax,␣-4(%%esi,␣%%ecx,␣4)\n\t"
        "movl␣%%ebx,␣(%%esi,␣%%ecx,␣4)\n\t"
        "decl␣%%ecx\n\t"
        "testl␣%%ecx,␣%%ecx\n\t"
        "jz␣set_to_one\n\t"
        "jmp␣start_loop\n\t"

        "no_swap:\n\t"
        "incl␣%%ecx\n\t"
        "jmp␣start_loop\n\t"

        "set_to_one:\n\t"
        "movl␣$1,␣%%ecx\n\t"
        "jmp␣start_loop\n\t"

        "end_sort:␣\n\t"
        :
        : "S"(arr), "D"(n)
        : "eax", "ebx", "ecx", "memory");
    gettimeofday(&end, NULL);
    return (end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec
        - start.tv_usec);
}
```

۲-۱-۴ تحلیل داده ها



با توجه به نمودار رسم شده میتوانیم به این نتیجه برسیم که استفاده از inline assembly تاثیر بسیار خوبی در زمان اجرا میگذارد و همان طور که مشاهده میشود در داده های با اندازه بزرگتر تاثیر آن بیشتر به چشم می آید.

۲-۲ bubble sort

۲-۲-۱ روند کلی

در این پروژه ابتدا ورودی توابع را در یک فایل نوشته و سپس تابع را به زبان c مینویسیم و آرایه سورت شده و زمان اجرای کد را در یک فایل متنی مینویسیم .در مرحله بعدی این تابع را به صورت inline assembly مینویسیم و آرایه سورت شده و زمان اجرای کد را در یک فایل متنی مینویسیم . سپس نمودار های تحلیلی زمان اجرایی دو برنامه را کشیده و در انتها تحلیل میکنیم .

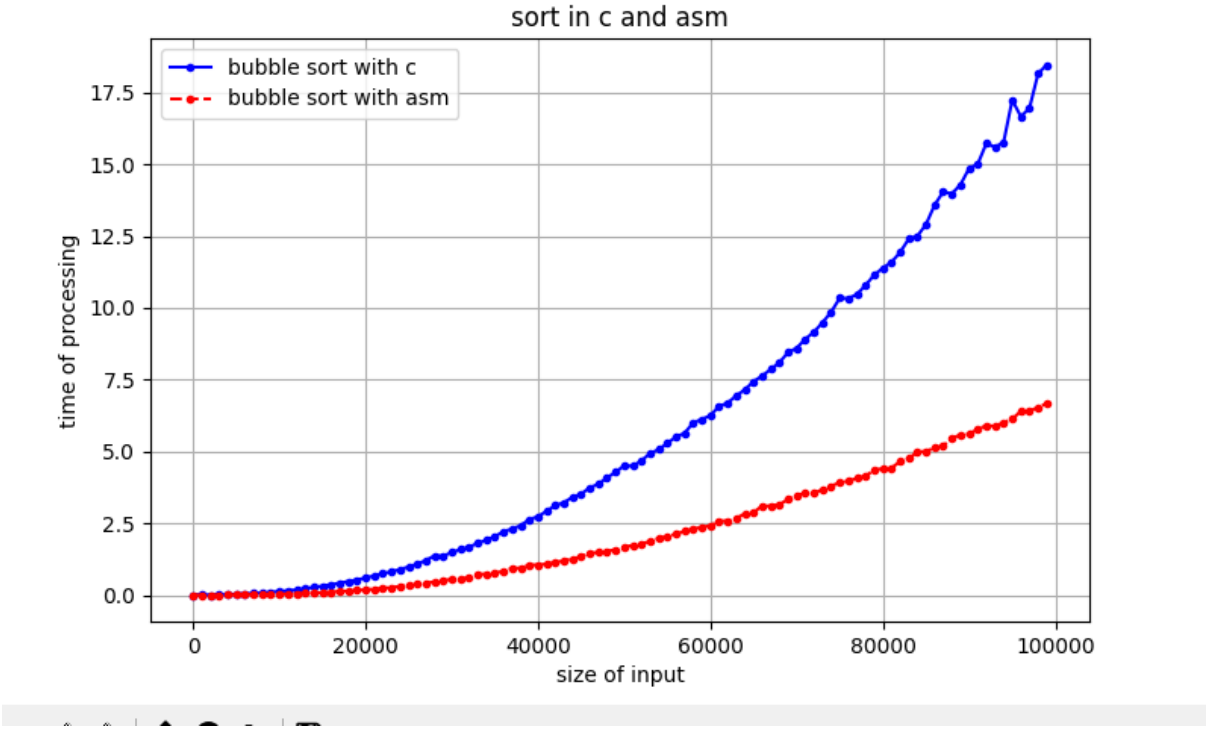
۲-۲-۲ تابع bubble Sort به زبان C

```c
long sort(int n, int *array)
{
    struct timeval start, end;
    gettimeofday(&start, NULL);

    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (array[j] > array[j + 1])
            {
                int temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }

    gettimeofday(&end, NULL);
    return (end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec
        - start.tv_usec);
}
```

۳-۲-۲ تابع bubble sort با به کارگیری inline assembly

```
long sort(int n, int *arr)
{
    struct timeval start, end;
    gettimeofday(&start, NULL);

    asm volatile(
        "movl␣$1,␣%%ecx␣\n\t"
        "start_loop:n\t"
        "cmpl␣%%edi,␣%%ecx\n\t"
        "jge␣end_sort\n\t"
        "movl␣(%%esi,␣%%ecx,␣4),␣%%eax\n\t"
        "movl␣-4(%%esi,␣%%ecx,␣4),␣%%ebx\n\t"
        "cmpl␣%%eax,␣%%ebx\n\t"
        "jle␣no_swap\n\t"
        "movl␣%%eax,␣-4(%%esi,␣%%ecx,␣4)\n\t"
        "movl␣%%ebx,␣(%%esi,␣%%ecx,␣4)\n\t"
        "decl␣%%ecx\n\t"
        "jnz␣start_loop\n\t"
        "jmp␣no_swap\n\t"
        "no_swap:\n\t"
        "incl␣%%ecx\n\t"
        "jmp␣start_loop␣\n\t"
        "end_sort:\n\t"
        :
        : "S"(arr), "D"(n)
        : "eax", "ebx", "ecx", "memory");

    gettimeofday(&end, NULL);
    return (end.tv_sec - start.tv_sec) * 1000000 + (end.tv_usec
        - start.tv_usec);
}
```

۲-۲-۴ تحلیل داده ها



با توجه به نمودار رسم شده میتوانیم به این نتیجه برسیم که استفاده از inline assembly تاثیر بسیار خوبی در زمان اجرا میگذارد و همان طور که مشاهده میشود در داده های با اندازه بزرگتر تاثیر آن بیشتر به چشم می آید

۲-۳ چالش های پروژه

۱- یادگیری و پیاده سازی روش inline assembly

۲ـ خواندن ورودی توابع با استفاده از فایل متنی و نوشتن زمان اجرا و توابع سورت شده درفایل متنی

۳- به دست آوردن نمودار زمان اجرا با استفاده از کد پایتون

فصل ۳

نتیجه‌گیری

پس از انجام این پروژه و تحلیل داده نتایج کلی زیر را به دست آوردیم و میتوان نتایج ان را به پروژه های در ابعاد بزرگ تر تعمیم دهیم :

۱ ـ به دلیل دسترسی مستقیم به دستورالعمل های پردازنده برنامه ای که با inline assembly نوشته میشود کارایی بالایی دارد.

۲ ـ به دلیل امکان مدیریت مستقیم رجیستر ها میتوانیم به دستورالعمل های خاصی دسترسی داشته باشیم که نمیتوان با اسفاده از زبان های سطح بالا به ان ها دسترسی پیدا کرد بنابراین استفاده از inline assembly میتواند زمان اجرا را تا حدودی بهبود دهد.

۳ ـ در این پروژه ما با ادغام زبان های سطح بالا توانستیم از مزایای اسمبلی بهره مند شویم بدون آنکه کل برنامه را به زبان اسمبلی بنویسیم.

۴ ـ به دلیل استفاده از inline assembly بخشی از کد که به زبان Assembly نوشته شده مستقیما در ماشین اجرا میشود و در نتیجه سربار کامپایلر کاهس می یابد.